
Microsoft®
.net™

Програмиране за .NET Framework

http://www.nakov.com/dotnet/

Сигурност в .NET Framework

Светлин Наков

Национална академия по разработка на софтуер

academy.devbg.org

Необходими знания

- Базови познания за .NET Framework
- Базови познания за езика C#
- Базови познания за работата на CLR, асемблита и атрибути

Съдържание

- Сигурността в .NET Framework
  - Безопасност на типовете и защита на паметта
  - Хващане на аритметични грешки
  - Симетрично и асиметрично кодиране. Цифров подпис
  - Силно-именувани асемблита
  - Технологията IsolatedStorage
- Code Access Security
  - Политиките за сигурност в .NET Framework
  - .NET Security Policy Editor
  - Права (Permissions)
  - Декларативно и програмно искане на права
  - "Stack Walk" и контрол над правата

Microsoft .net

Съдържание (2)

- Role-Based Security
  - Автентикация и авторизация
  - Identity и Principal обекти
  - Създаване на Identity и Principal обекти
  - Авторизация по Principal
- Криптография в .NET Framework
  - Изчисляване на хеш стойност
  - Подписване на XML (XMLDSIG)

Сигурността в .NET Framework

- .NET Framework е проектиран с мисията да бъде сигурна и надеждна платформа
  - Управляваният код (Managed Code) се изпълнява контролирано
    - CLR непрекъснато се грижи за сигурността
  - Компоненти, свързани със сигурността:
    - Type checker – проверка на типовете
    - Exception manager – управление на изключенията
    - Security engine – управление на сигурността на кода (Code Access и Role-Based Security)
  - Сигурността в .NET разширява и допълва сигурността в Windows

Безопасност на типовете

- Управляваният код е защитен от неправилна работа с типовете
  - Не се използват указатели към паметта
  - Използват се референции към обекти
    - Референциите са силно типизирани
    - Не можем да присвоим на референция към обект референция към несъвместим обект

```
object bytes = new byte[5];
char[] chars = (char[]) bytes;
// System.InvalidCastException is thrown
```

  - Ограничава се достъпът до чужди обекти и региони от паметта
  - Решен е проблемът "buffer overrun"

Проблемът "Buffer overrun"

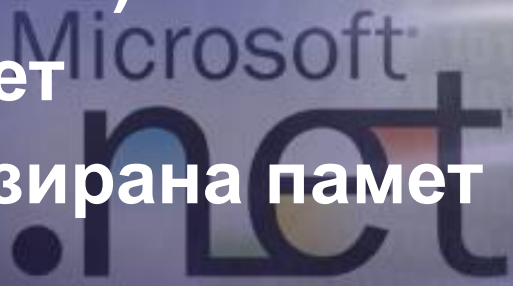
- В .NET Framework масивите и символните низове не могат да се препълнят:

```
private static void CopyArray(byte[] src,
    byte[] dest, int size)
{
    for (int i=0; i<size; i++)
        dest[i] = src[i];
}

static void Main()
{
    byte[] arr1 = new byte[10];
    byte[] arr2 = new byte[5];
    CopyArray(arr1, arr2, 10);
    // System.IndexOutOfRangeException is thrown
}
```

Microsoft .net

Защита на паметта

- CLR автоматично управлява паметта
  - Динамично-заделените обекти се разполагат в т. нар. Managed heap
  - Неизползваните обекти се почистват автоматично от т. нар. Garbage Collector
- Някои от най-неприятните проблеми в програмирането са почти невъзможни:
  - Загуба на памет (memory leaks)
  - Достъп до освободена памет
  - Използване на неинициализирана памет



Хващане на аритметични грешки

- При работа с аритметични операции са възможни препълвания на типовете
  - При получаване на резултат, който не се събира в типа, който е използван
  - При преобразуване на типове
- В .NET Framework има вграден механизъм за прихващане на аритметични препълвания за целочислените типове
- В C# се използва ключовата дума `checked`

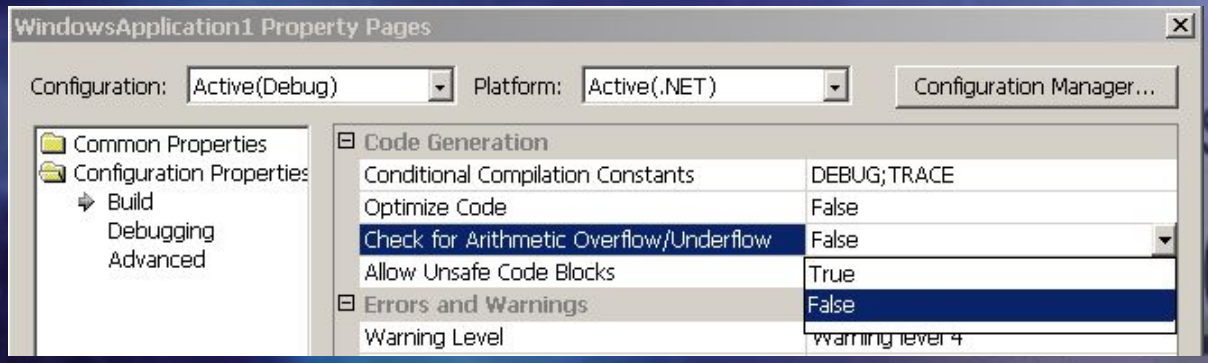
```
checked
{
    int square = a*a;
}
```

Microsoft .net

Хващане на аритметични грешки

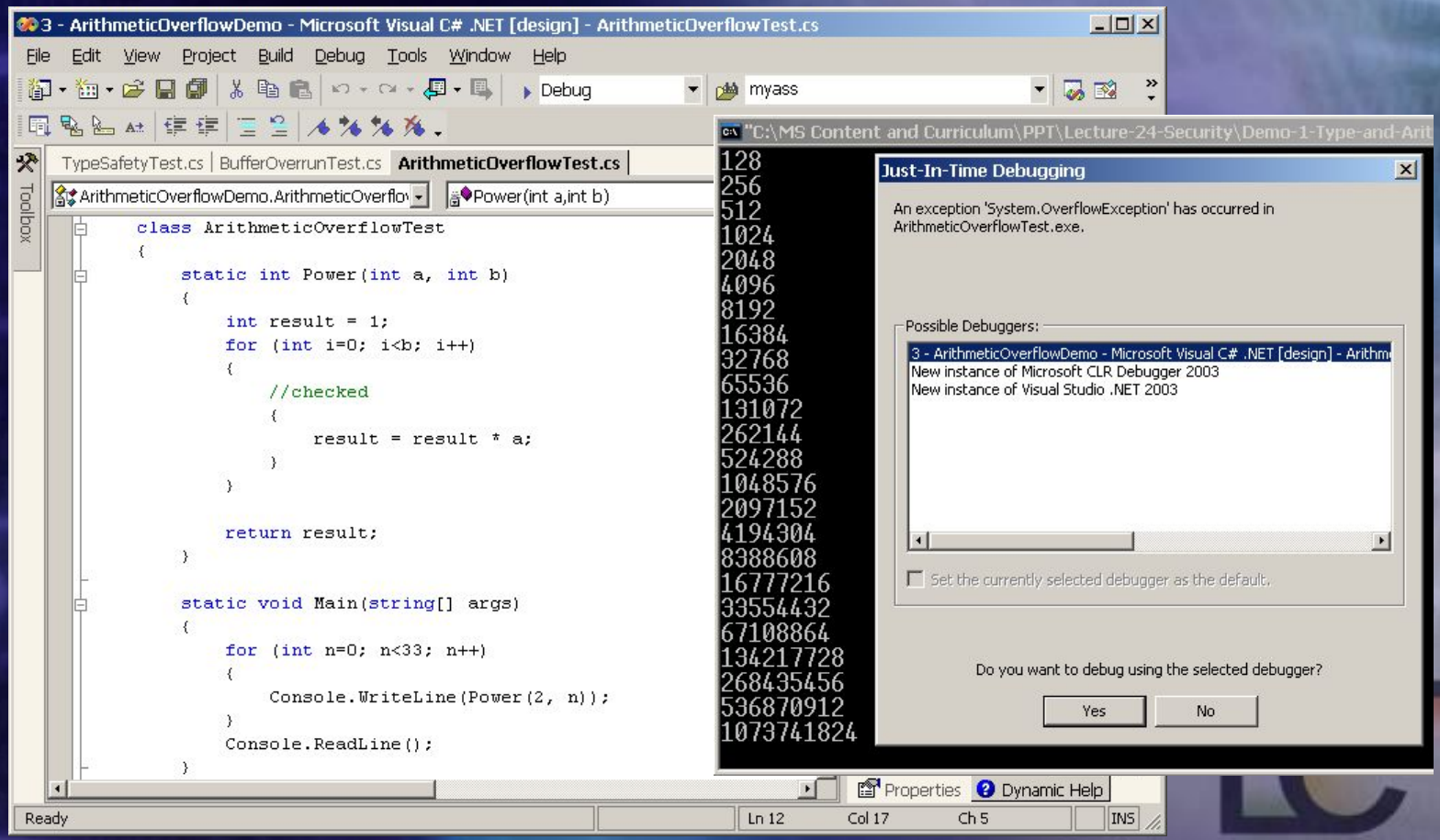
- Ключови думи `checked` и `unchecked`
  - Включват / изключват проверката за препълване за даден програмен блока
- По подразбиране проверката за препълване е изключена
- Може да се включва/изключва с опции на C# компилатора и с настройки във VS.NET:

```
csc /checked+ SomeFile.cs
```



Демонстрация #1

- Безопасност на типовете и аритметичните операции

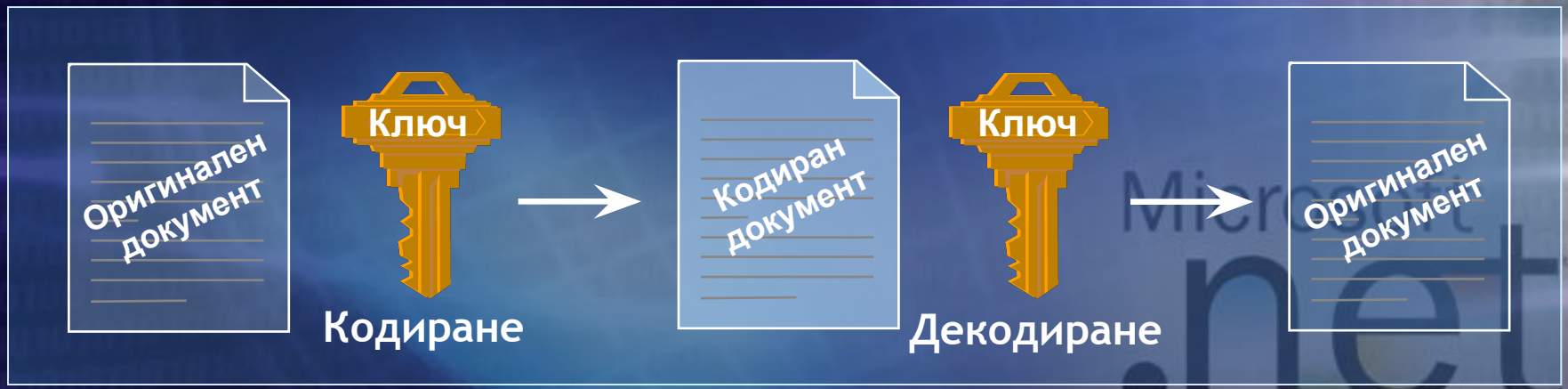


Application Domains

- Application Domains:
  - Позволяват няколко .NET приложения да работят в един процес на ОС
  - Application Domains са изолирани един от друг по памет, данни и код
  - Подобрена производителност
    - Не е необходима междупроцесна комуникация при обмяна на данни
  - Повишена сигурност при изпълнение на дадено асембли
    - Може фино да се настройва
  - Възможно е стартиране на .NET приложения с ограничени права

Симетрично кодиране

- Симетричните кодиращи схеми:
  - Кодират и декодират с един и същ ключ
  - Работят бързо
  - Могат да обработват потоци от данни
  - Алгоритми: DES, 3DES, RC2, RC4, IDEA
  - Кодиране и декодиране:



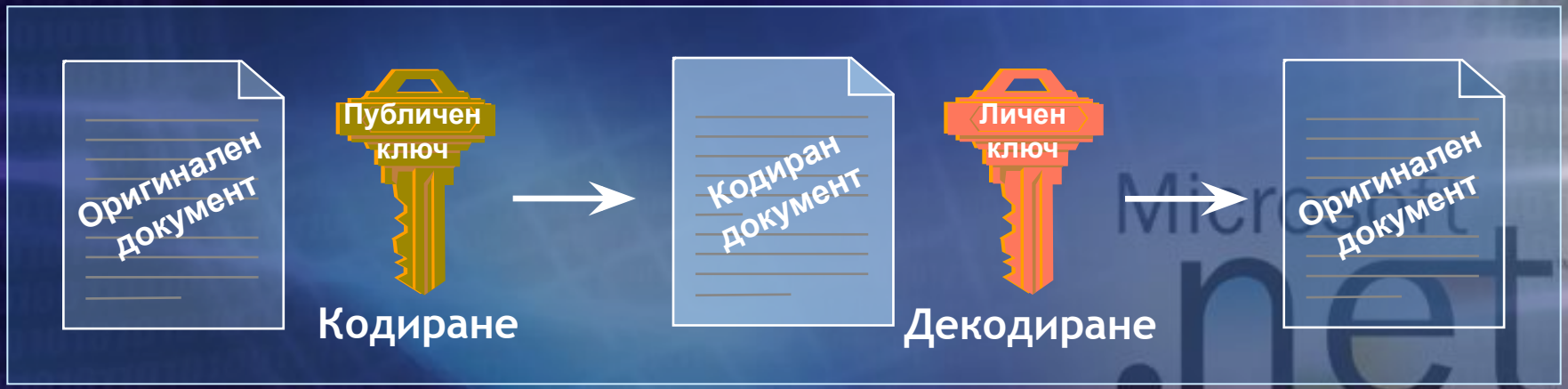
Асиметрично кодиране

- Криптографията с публични ключове (Public Key Cryptography):
  - Използва асиметрични ключове
    - Двойка съответни публичен / личен ключ (public/private key pair)
    - От единия ключ не може да се извлече другият
  - Работи много по-бавно от симетричната криптография
  - Не работи добре с потоци от данни
  - Алгоритми: RSA, DSA, Diffie-Hellman, ECDSA (Elliptic-Curves DSA)

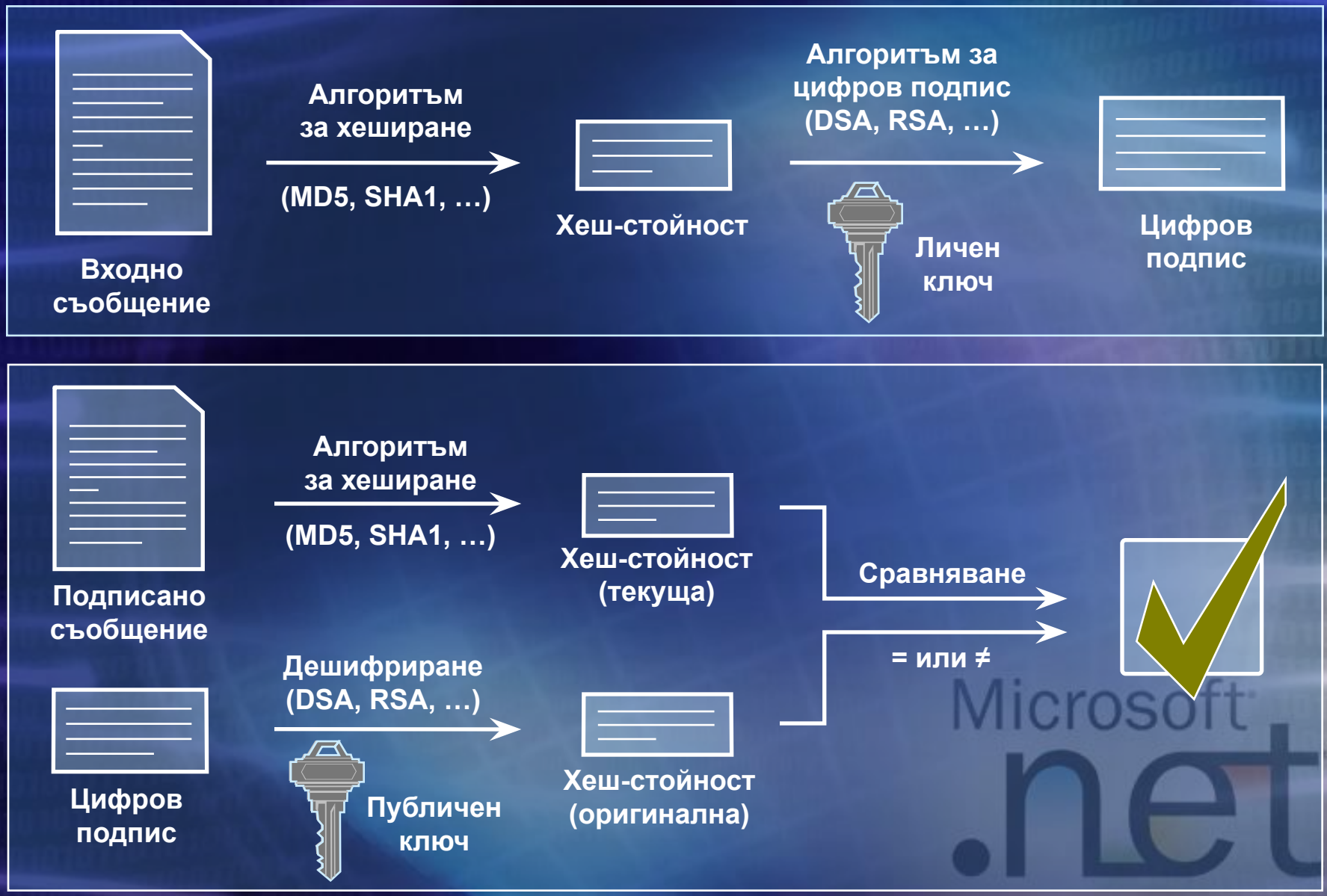
Microsoft .net

Асиметрично кодиране

- Криптографията с публични ключове:
  - Може да използва инфраструктурата на публичния ключ (PKI)
    - Т. нар. сертифициращи организации чрез цифрови сертификати гарантират че даден публичен ключ е свързан с дадено лице
  - Кодиране и декодиране:



Цифров подпис



Силно-именувани асемблита

- Силното име на асембли:
  - Уникално идентифицира асемблито
  - Съдържа цифров подпис и съответен на него публичен ключ
  - Състои се от:
    - Име (напр. `System.Windows.Forms`)
    - Версия (напр. `1.0.5000.0`)
    - Култура (напр. `neutral`)
    - Публичен ключ (напр. `b77a5c5...4e089`)
    - Цифров подпис
      - Съответства на публичния ключ
      - Проверява се от CLR при зареждане на асемблито

Силно-именувани асемблита

- Силно-именуваните асемблита:
  - Не позволяват промяна / подправяне
  - Потвърждават идентичността на производителя
  - Позволяват няколко версии на едно и също асембли да се инсталират и използват независимо
- Генериране на двойка ключове:

```
sn –k MyKeyPair.snk
```

- Подписване на асембли:

```
[assembly: AssemblyKeyFile(@"..\..\MyKeyPair.snk ")]
```

Силно-именувани асемблита

- Силно-именуваните асемблита могат да се инсталират в Global Assembly Cache:
  - Инсталиране и деинсталиране от GAC:

```
gacutil –i MyAssembly.dll
```

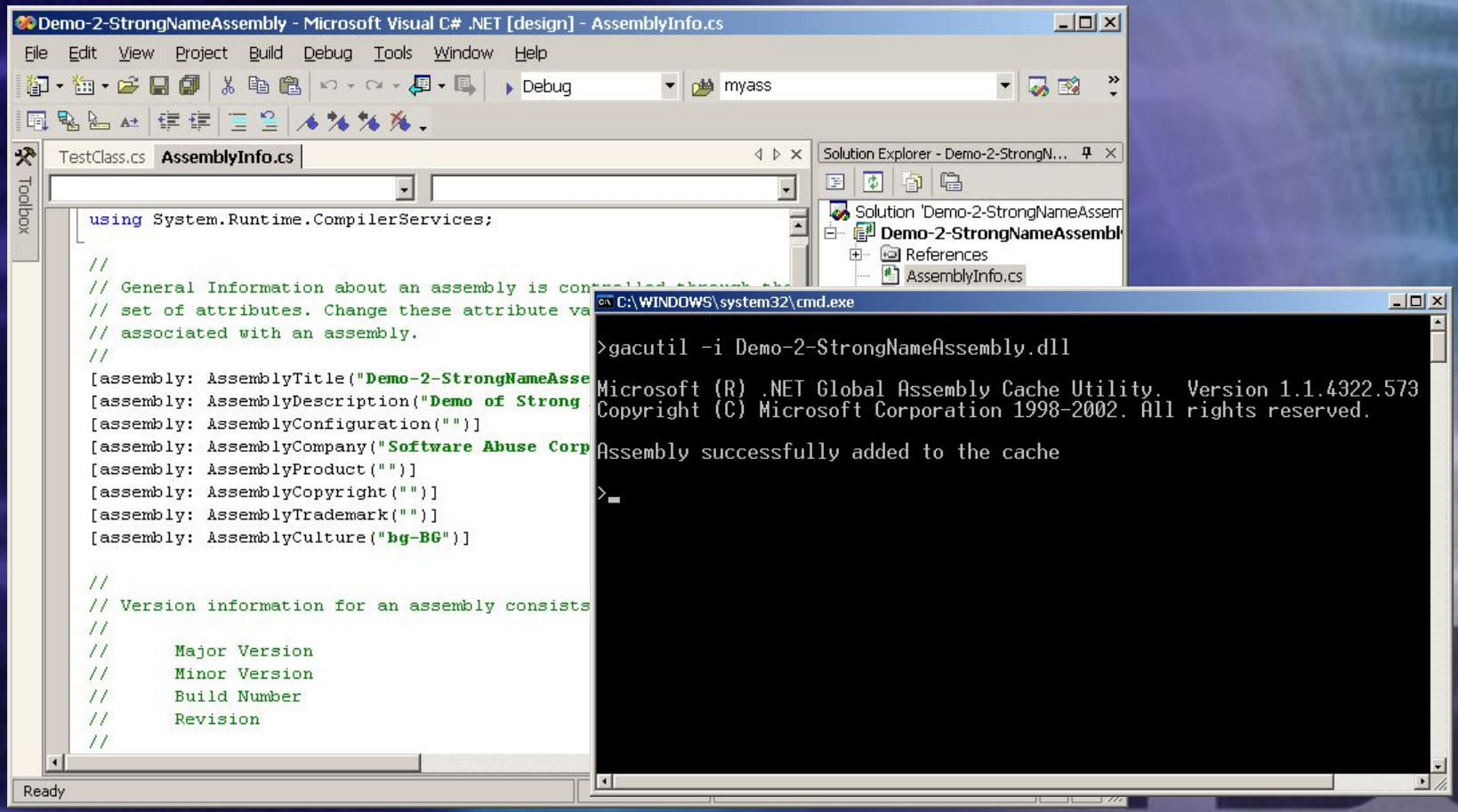
```
gacutil –u MyAssembly
```

- При добавяне на референция от някое асембли (`A.dll`) към силно-именувано асембли (`S.dll`):
  - Публичният ключ на `S.dll` се записва в компилираното асембли `A.dll`
  - Така `A.dll` се свързва само с конкретната версия на асемблито `S.dll`

Microsoft .net

Демонстрация #2

- Създаване на асембли със силно име и инсталиране в GAC



Технологията `IsolatedStorage`

- Технологията `IsolatedStorage`:
  - Позволява приложение, което няма права за достъп до локалния твърд диск, да съхранява работни файлове
- `IsolatedStorage` предоставя изолирана виртуална файлова система:
  - ограничена по обем
  - без да позволява на приложението достъп до останалите файлове на твърдия диск
- Често се използва от приложения, работещи с намалени права:
  - Например: Windows Forms контроли, стартирани от Web-страница в Интернет

Microsoft .net

Технологията IsolatedStorage

- Хранилищата за данни (IsolatedStorage) могат да имат обхват:
  - `IsolatedStorageScope.User` – хранилището е за текущия потребител
  - `IsolatedStorageScope.Assembly` – хранилището е за текущото асембли
  - Обхватите могат да се комбинират
- Получаване на `IsolatedStorage`:

```
IsolatedStorageFile store =
    IsolatedStorageFile.GetStore(
    IsolatedStorageScope.User |
    IsolatedStorageScope.Assembly,
    null, null);
```

Microsoft .net

Технологията IsolatedStorage

- Отваряне на файл от IsolatedStorage:

```
IsolatedStorageFileStream stream =
    new IsolatedStorageFileStream(
    "notes.txt", FileMode.Open,
    FileAccess.Read, isoStore);
```

- Местоположение на файловете:

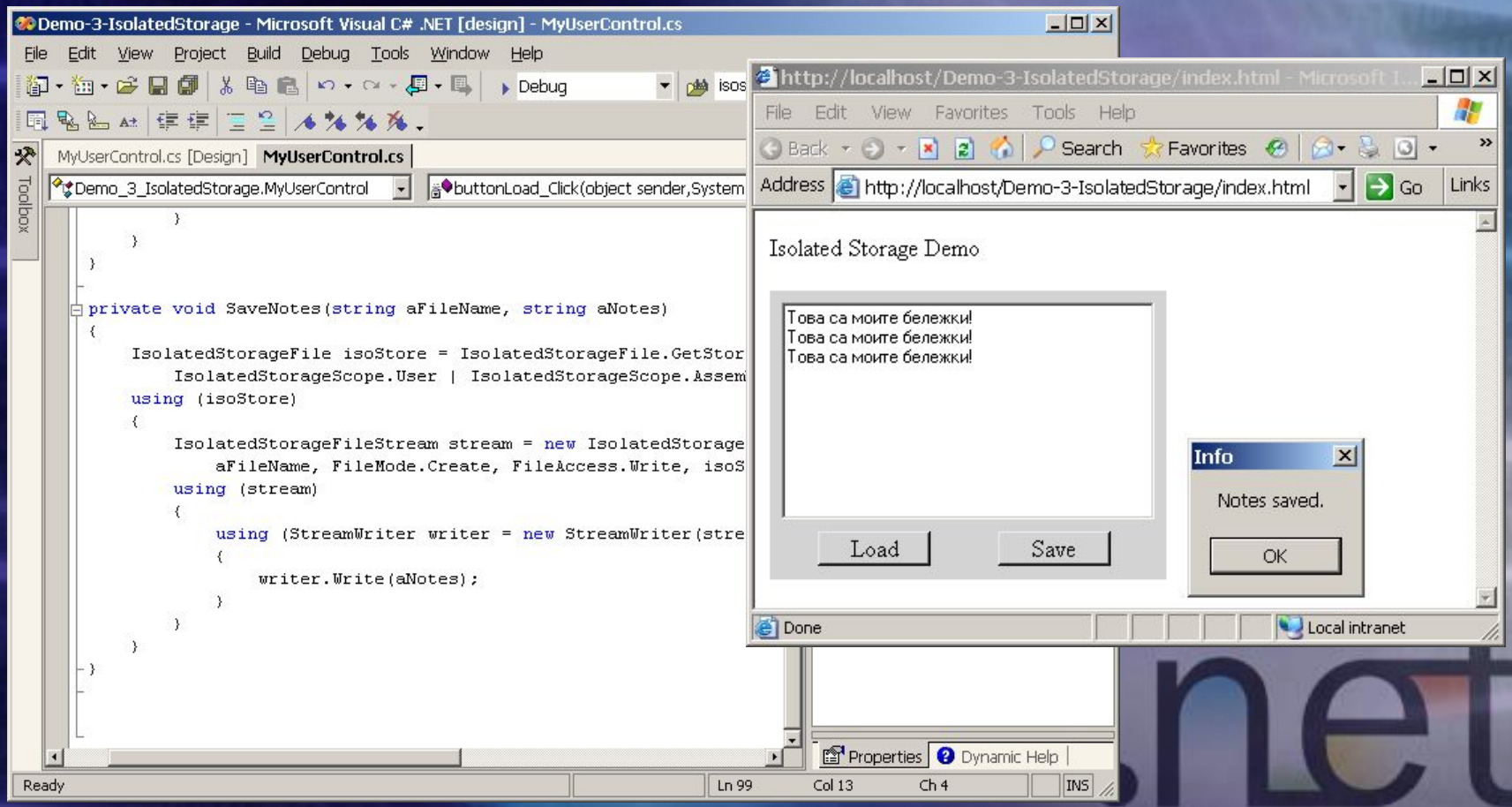
```
C:\Documents and Settings\<username>\Local
Settings\Application Data\IsolatedStorage\...
```

- Операции с файлове и директории:
  - `GetDirectoryNames()`, `GetFileNames()`, `DeleteFile(…)`, `CreateDirectory(…)`, `DeleteDirectory(…)`

Microsoft .net

Демонстрация #3

- Работа с `IsolatedStorage` от потребителска контрола в IE



Сигурност на кода

- Концепцията "сигурност на кода" (Code Access Security) е фундаментален принцип при дизайна на .NET Framework
  - Надгражда системата за сигурност на операционната система
- Сигурността се управлява от:
  - политиката за сигурност (.NET Framework Security Policy) чрез
    - права (permissions)
    - доказателства за произход (evidences)
- Политиката задава с какви права се изпълнява всяко асембли според неговите доказателства за произход

Сигурност на кода

- Code Access Security позволява CLR да изпълнява програмен код с ограничени права, по-ниски от правата на потребителя
- За всяко асембли CLR събира съвкупност от доказателства за произход (evidences)
  - Силно име на асембли
  - URL, от където идва асемблито
  - Интернет зона, от където идва асемблито
  - Authenticode цифров подпис
  - Хеш-код на асемблито
- CLR не може да даде на никое асембли права, по-високи от правата на текущия потребител в ОС

Политиките за сигурност в .NET

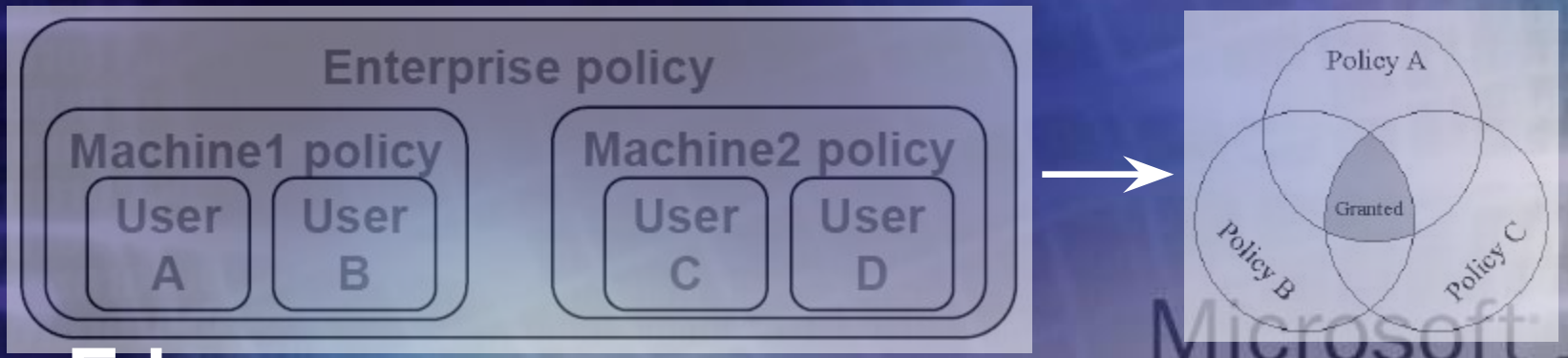
Термин	Описание

Именуван списък с права (Permission Set)	• Представлява именуван списък с права
	• Примери:
	  • **FullTrust (всички права)**
	  • **[User Interface; Printing; Web Access]**

Група код (Code Group)	• Базира се на доказателства • Обединява асемблита с еднакви доказателства, напр. всички асемблита, идващи от http://www.devbg.org/ • Групите се наследяват йерархично
Политика за сигурност (Security Policy)	• Задава се от администраторите • Прилага се по време на изпълнение • Дефинира именуваните списъци с права (Permission Sets) • Дефинира групите код (Code Groups) • Задава списък с права за всяка група код (свързва правата с групите)

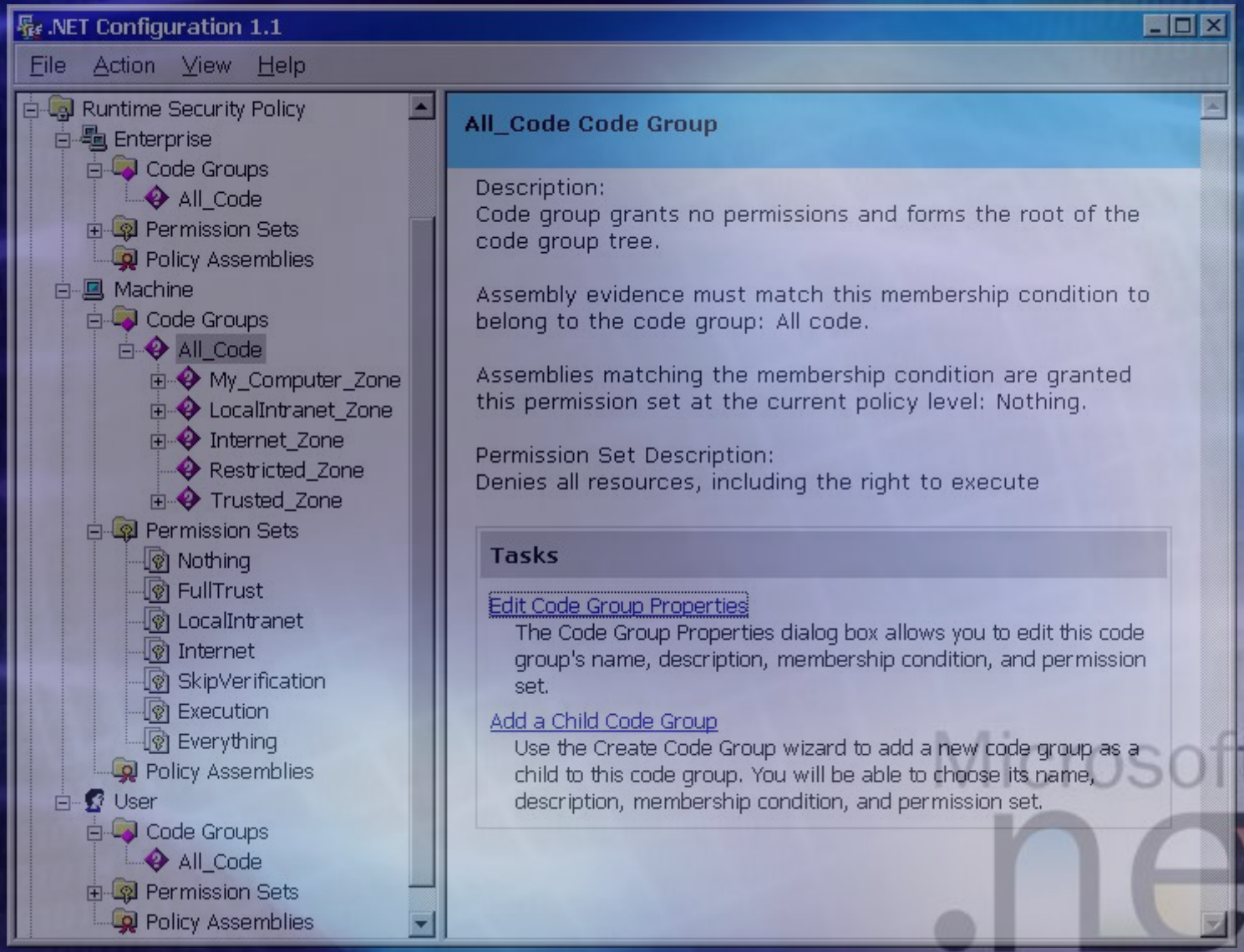
Нива на политиките за сигурност

- Има няколко нива на политиките:
  - Enterprise: политика за организацията (за Windows домейна)
  - Machine: политика за всички потребители на машината
  - User: политика за текущия потребител



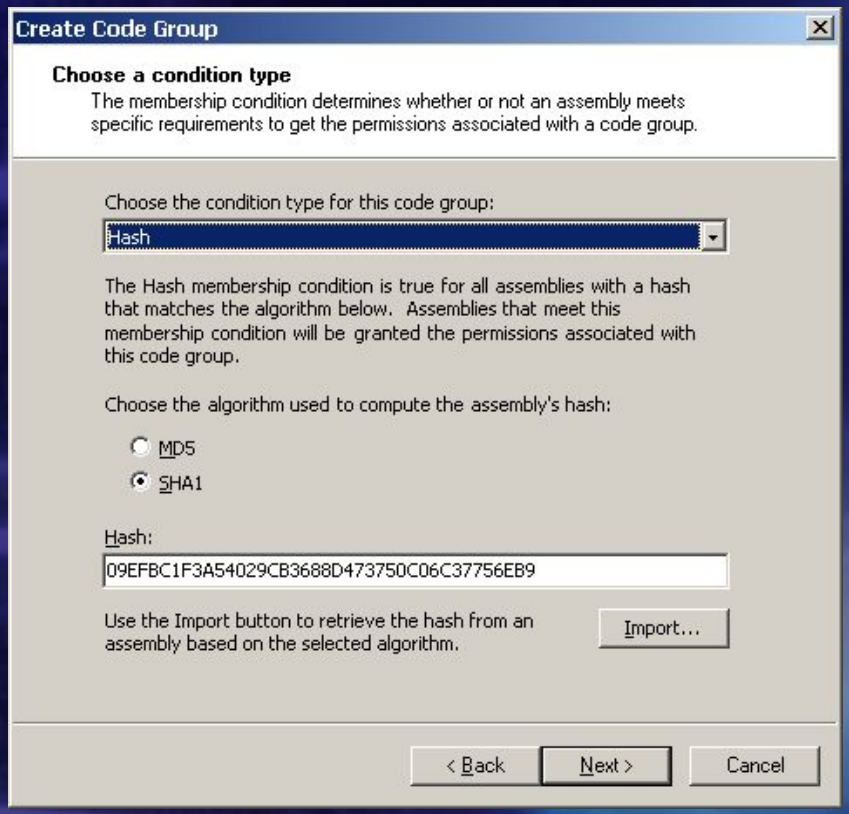
- Ефективната политика е сечението на правата, дадени от трите нива

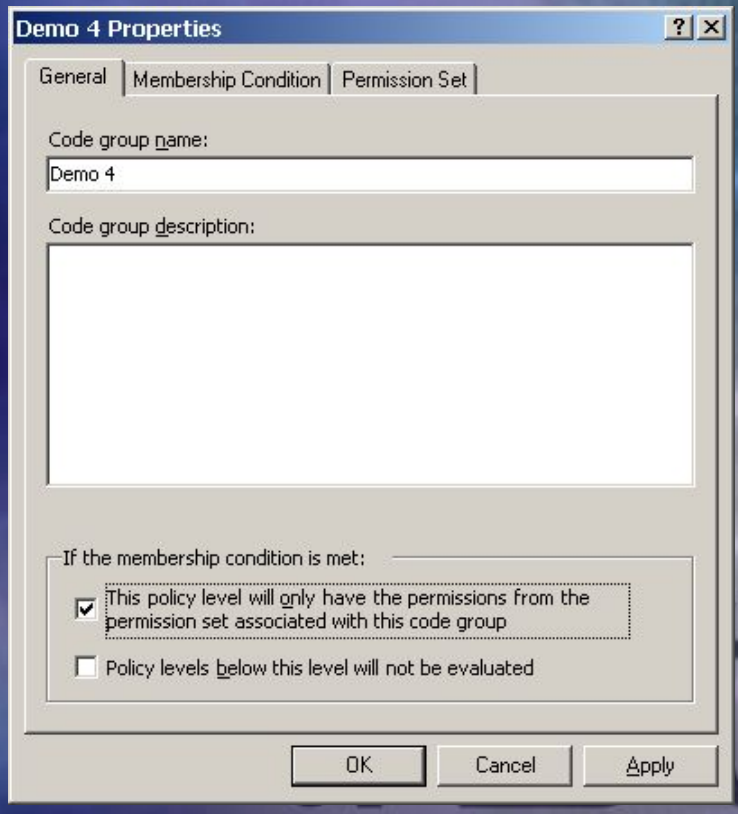
.NET Security Policy Editor



Демонстрация #4

- .NET Security Policy Editor – настройка на правата на дадено асембли по SHA1 хеш код





Права (Permissions)

- Permission обектите представляват специфично право (разрешение) за достъп до ресурс или извършване на някаква операция, например:
  - право за печатане на принтер
  - право за достъп до SQL Server
- Даването на право (permission grant) разрешава на дадено асембли (код) да го използва
- Изискването на право (permission demand) проверява дали извикващият код има дадено право

Стандартни .NET права

- .NET Framework дефинира стандартни класове за правата за достъп до ресурси:

Право	Описание
FileIOPermission	Четене / писане по файловата система
IsolatedStorageFilePermission	Достъп до изолирана виртуална файлова система тип "IsolatedStorage"
UIPermission	Използване на Windows Forms GUI
FileDialogPermission	Достъп до диалога за избор на файл
PrintingPermission	Печатане на принтер
WebPermission	Достъп до Web ресурси
SocketPermission	Работа със сокети
OleDbPermission, SqlClientPermission	Достъп до база данни през OleDb или SQLClient Data Provider-ите
RegistryPermission	Достъп до Windows Registry
ReflectionPermission	Достъп до Reflection

Декларативно искане на права

- Искането на право (permission request)
  - Използва се за да се укаже какви права изисква дадено асембли, метод или фрагмент от кода, за да работи нормално
  - Задава се декларативно – чрез атрибути:

```
[assembly: PrintingPermission(
  SecurityAction.RequestMinimum,
  Level=PrintingPermissionLevel.SafePrinting)]
```

  - Ако поисканото право не може да бъде дадено, се хвърля `SecurityException`
  - Ако поисканото право за дадено асембли не бъде дадено, асемблито не се зарежда
  - Използва се най-често от клиентски код

Декларативно искане на права

- При декларативното искане на права се задава параметър `SecurityAction`:

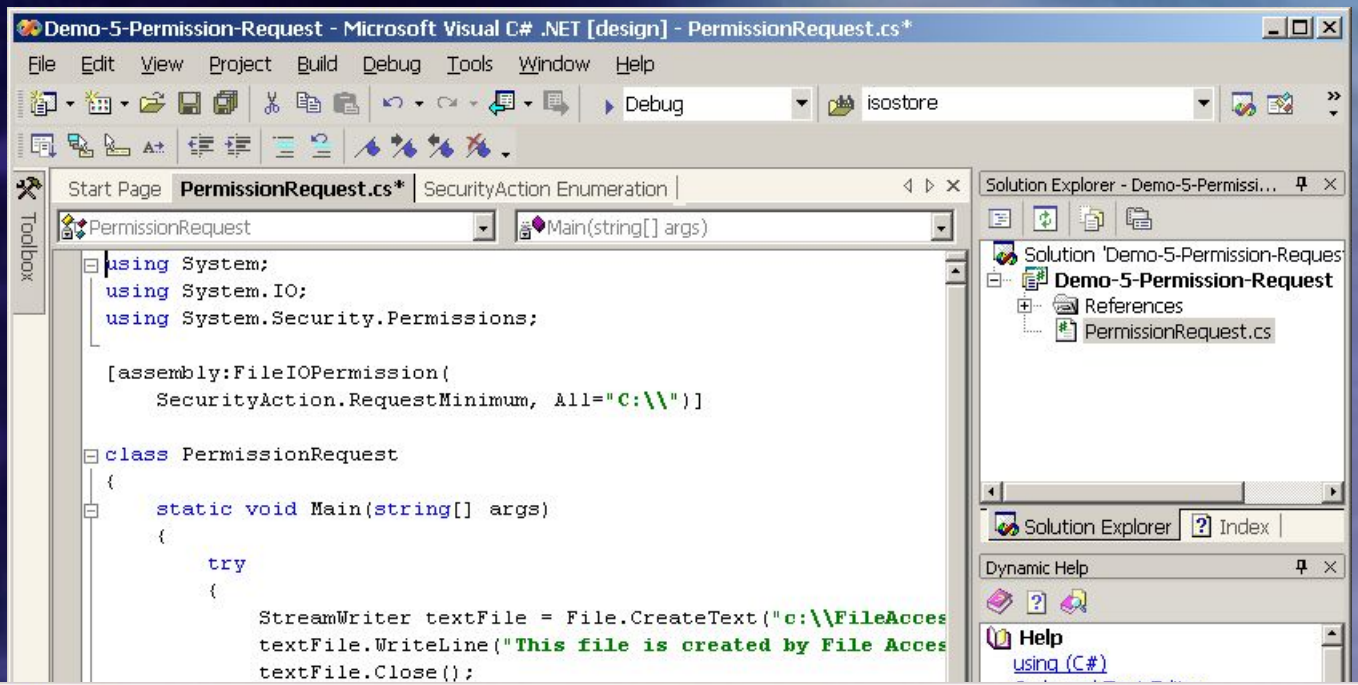
```
[assembly:FileIOPermission(
    SecurityAction.RequestRefuse, All="C:\\")]
```

  - `RequestMinimum` – указва, че асемблито не може да работи без съответното право
  - `RequestRefuse` – указва, че асемблито иска зададеното право да му бъде отнето
  - `Demand` – указва, че всички асемблита от стека на извикване трябва да имат зададеното право
  - `Assert`, `Deny`, `PermitOnly` – отнасят се за класове и методи
    - Управляват сигурността на извиквания код чрез контролиране на т. нар. "Stack Walk"

Microsoft .net

Демонстрация #5

- Декларативно искане на права



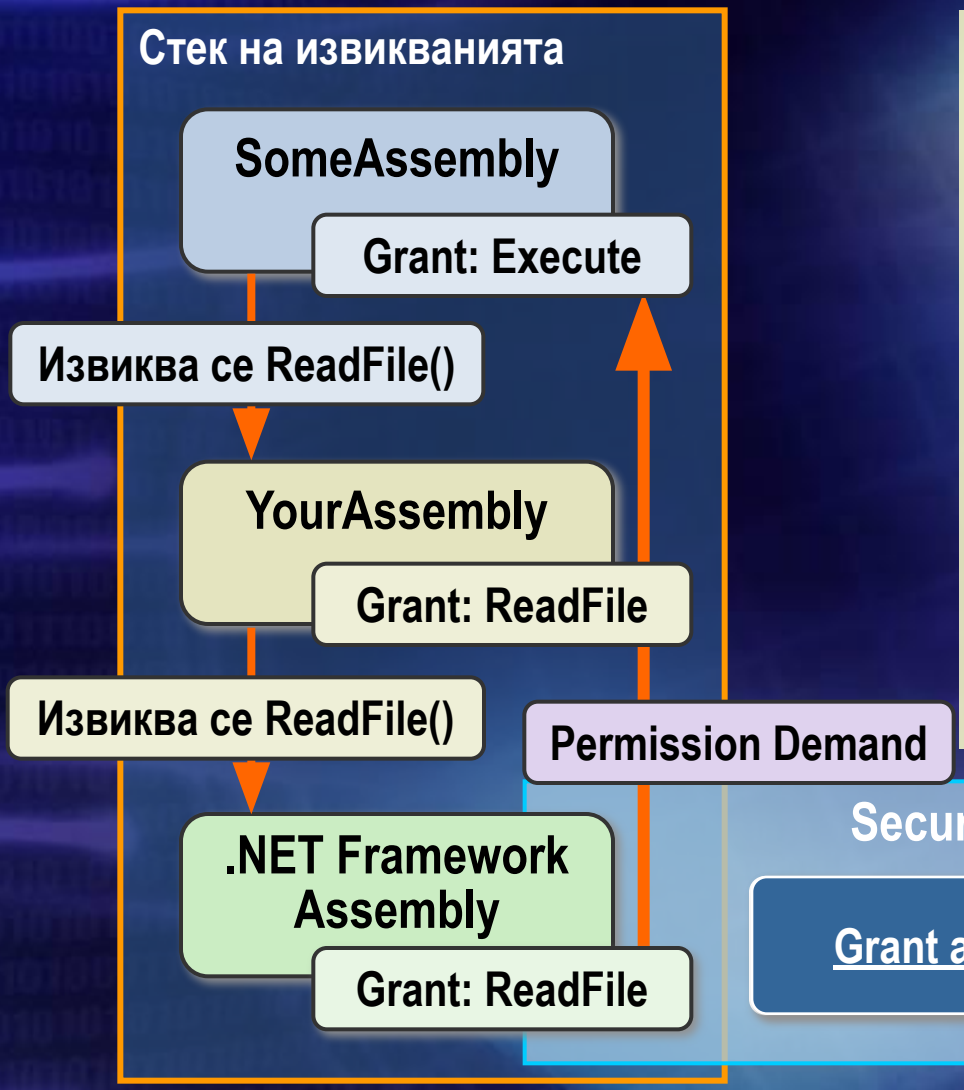
Програмно искане на права

- Програмното искане на права
  - Позволява даден код да поиска дадено право по време на изпълнение
  - Използва се най-вече при разработка на библиотеки с класове и сървърски код
  - Пример:

```
FileDialogPermission fdp = new FileDialogPermission(
    PermissionState.Unrestricted);
fdPerm.Demand();
```

  - Методът `Demand()` проверява дали текущото асембли и всички извикващи го асемблита по стека имат поисканото право
    - Предизвиква изключение при неуспех

Какво е "Stack Walk"?

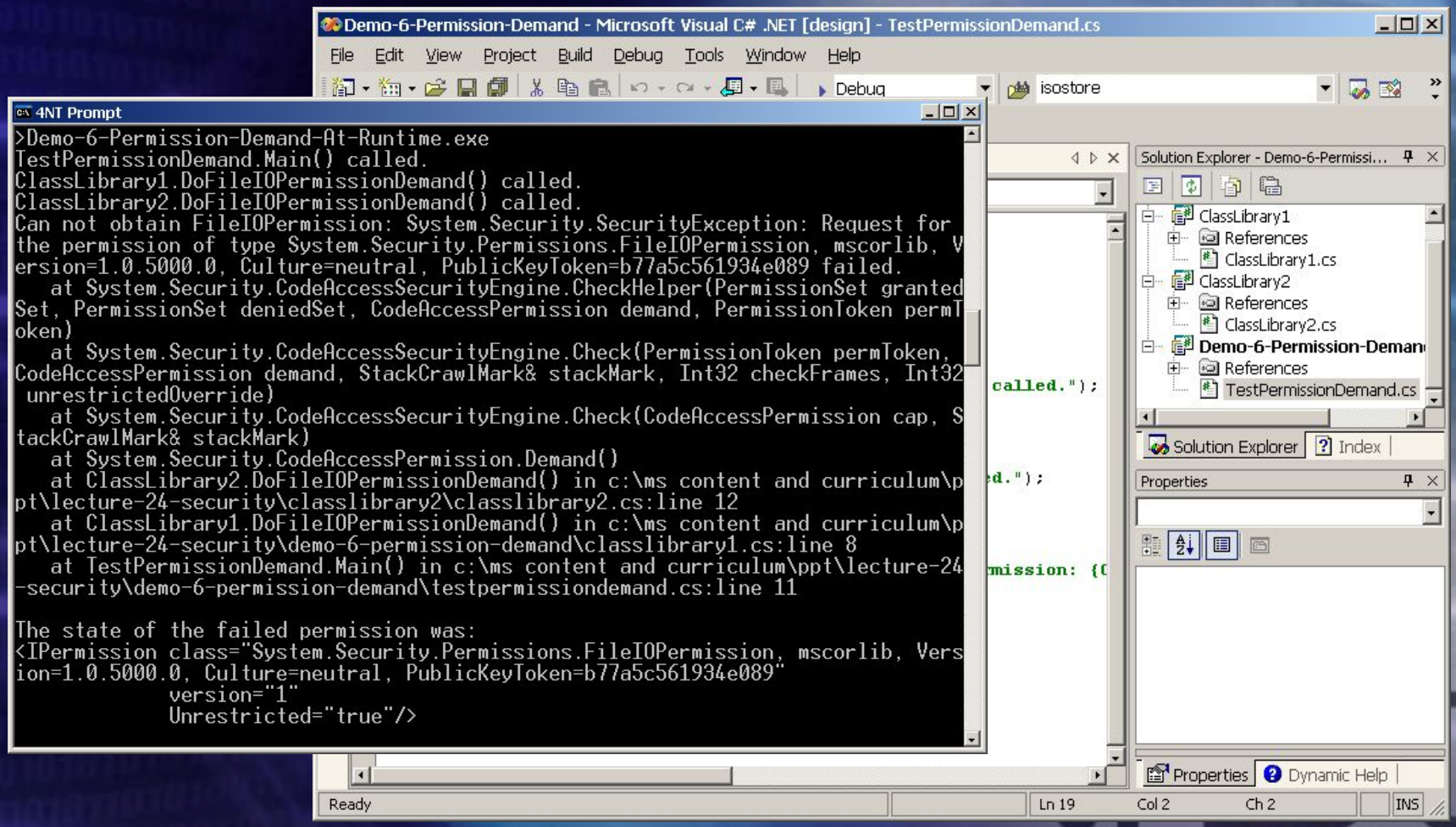


1. Някое асембли иска достъп до метод в нашето асембли
2. Нашето асембли препраща заявката до някое системно асембли от .NET Framework
3. Системата за сигурност проверява дали всички асемблита в стека имат необходимите права и съответно дава достъп
4. При неуспех системата за сигурност хвърля изключение

Microsoft .net

Демонстрация #6

- Програмно искане на права



Контрол над правата

- Правата за достъп до кода (code-access permissions) имат "Stack Walk" семантика
  - За да се даде дадено право на едно асембли, трябва всички извикващи асемблита по стека също да го имат
  - "Stack Walk" механизмът не позволява асембли с ниски права да извършва непозволени действия през асембли с високи
- Възможно е програмно въздействие върху "Stack Walk" процедурата:
  - надолу по стека – дали да се проверяват правата на извикващите методи
  - нагоре по стека – какви права да се дават на извиканите методи

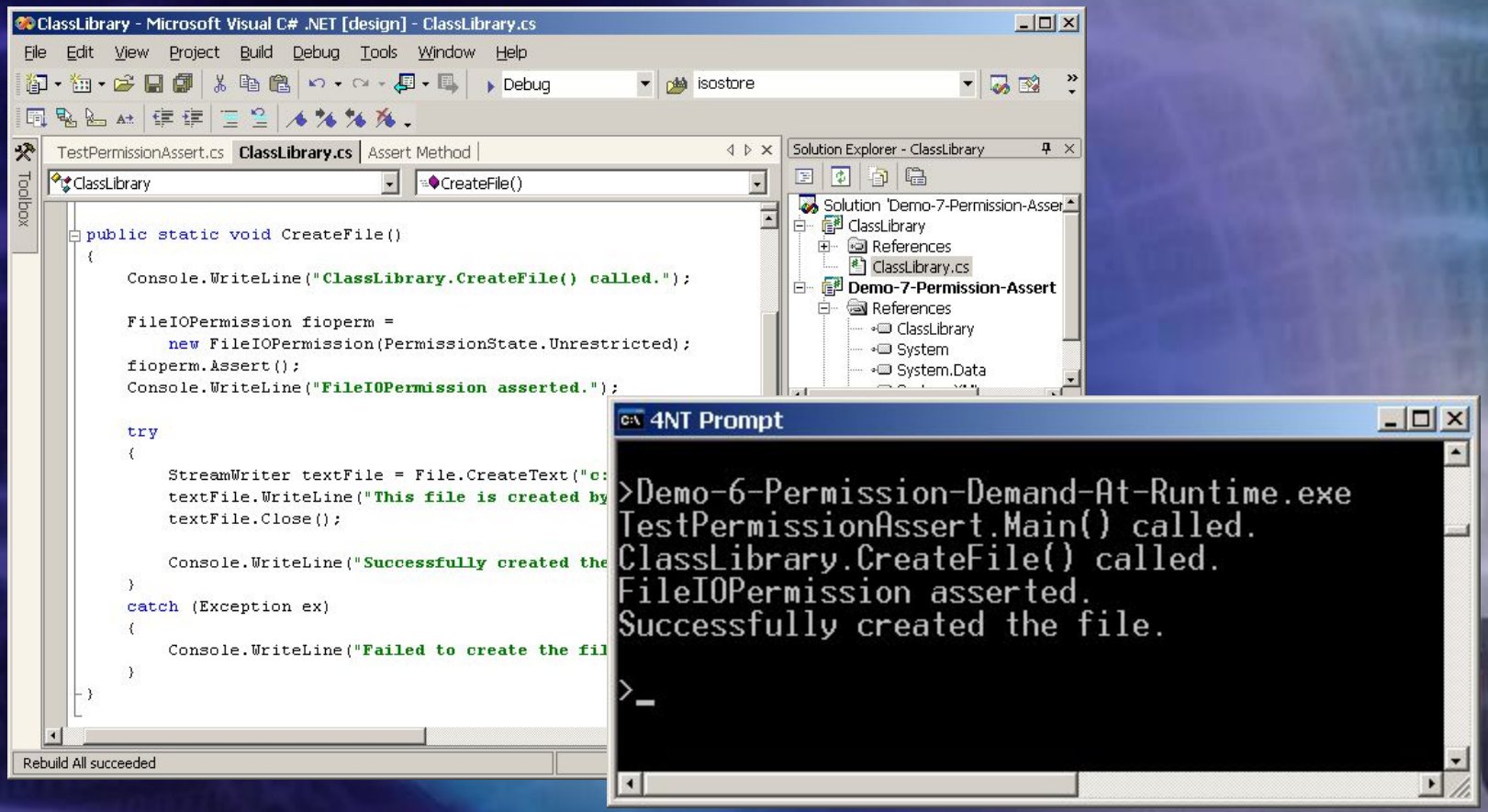
Microsoft .net

Контрол над правата

- При писане на компоненти се налага едно асембли да ползва временно някои от правата на друго
- Следните методи контролират "Stack Walk" процедурата
  - `Assert()` – позволява текущия метод да ползва пълните права на асемблито си, независимо от правата на извикващите го асемблита
  - `Deny()` – временно отнема даденото право за всички методи извикани от текущия
  - `PermitOnly()` – временно отнема всички права без даденото за всички методи извикани от текущия

Демонстрация #7

- Контрол над "Stack Walk" процедурата чрез `Assert()`



Сигурност базирана на роли

- Сигурност, базирана на роли (Role-Based Security)
  - Стандартно средство за контрол на достъпа в .NET Framework
  - Може да се използва програмно и декларативно
  - Базира се на потребители и роли (Identities и Principals)
  - Всяка нишка в .NET Framework си има потребител и роля, свързани с нея

Автентикация и авторизация

- Автентикация (authentication)
  - Процесът на проверка дали даден потребител е този, за който се представя
  - Може да става с парола, със сертификат, със смарт-карта или по друг начин
- Авторизация (authorization)
  - Процесът на проверка дали даден потребител има право да извърши дадено действие
  - Предполага се че потребителят е успешно автентикиран
  - Role-Based Security осигурява механизми за авторизация в .NET приложенията

Microsoft .net

Identity и Principal обекти

- Identity (самоличност) – потребител
  - Съдържа информация за потребителя, в контекста на който се изпълнява кода:
    - Съдържа: име на потребител, домейн, дали е автентикиран и др.
- Principal (принципал) – роли
  - Съдържа контекста за сигурност на потребителя, в който се изпълнява кода
  - Съдържа ролите, в които потребителят участва
  - Съдържа Identity информацията за потребителя

Microsoft .net

Identity и Principal обекти

- В .NET Framework има два типа Identity и Principal обекти:
  - `WindowsIdentity` и `WindowsPrincipal`
    - Свързани са с потребителите и техните роли в контекста на Microsoft Windows
    - Съдържат Windows специфична информация
  - `GenericIdentity` и `GenericPrincipal`
    - Дават възможност за организиране на собствени системи за авторизация
    - Съдържат информация, зададена от програмиста, която не е обвързана с Windows

Microsoft .net

WindowsPrincipal – пример

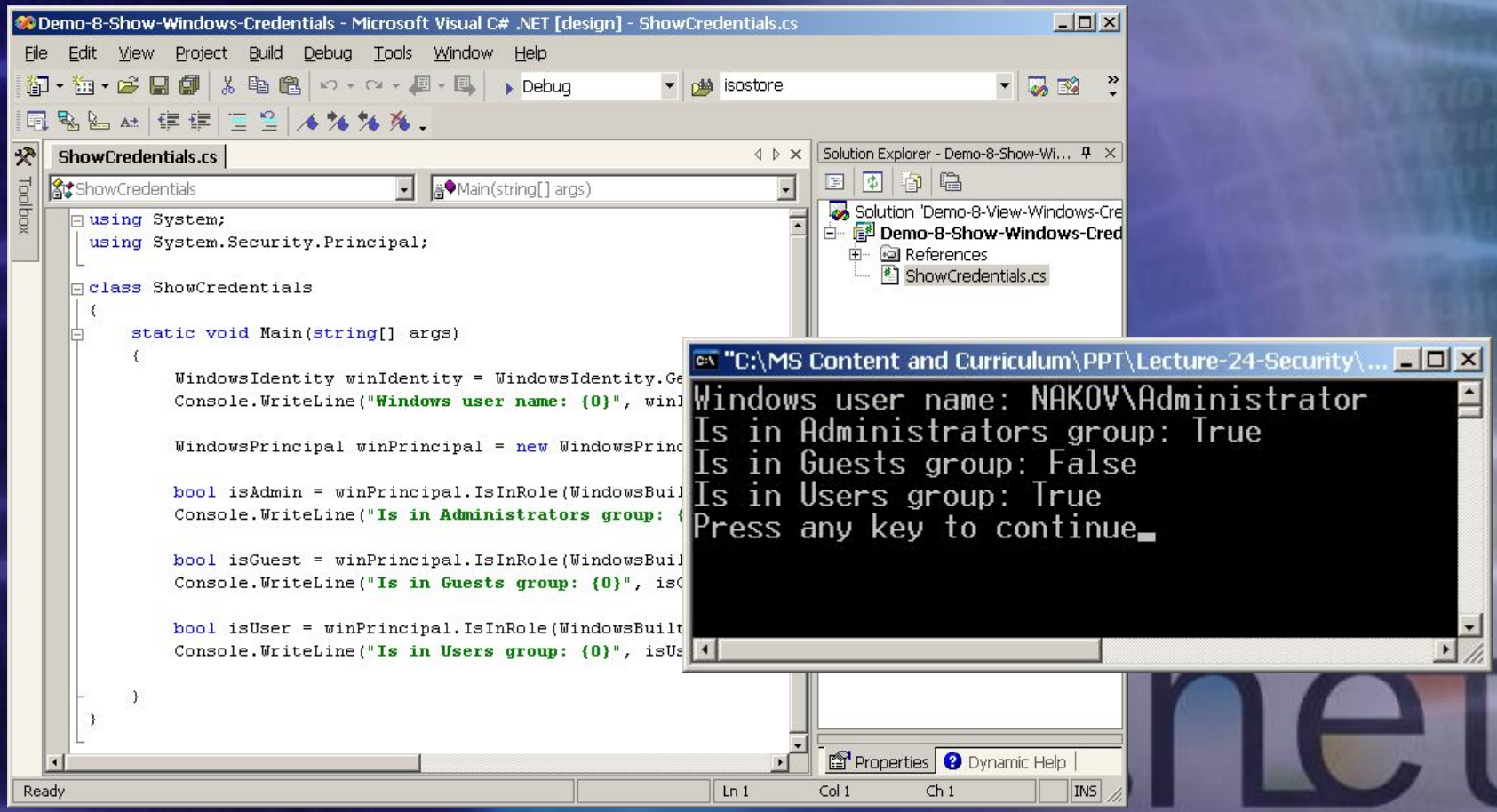
- Извличане на текущия Windows потребител и информация за него:

```
WindowsIdentity winIdentity =
    WindowsIdentity.GetCurrent();
Console.WriteLine("Windows login: {0}",
    winIdentity.Name);

WindowsPrincipal winPrincipal =
    new WindowsPrincipal(winIdentity);

bool isAdmin = winPrincipal.IsInRole(
    WindowsBuiltInRole.Administrator);
Console.WriteLine("Administrator: {0}", isAdmin);

bool isGuest = winPrincipal.IsInRole(
    WindowsBuiltInRole.Guest);
Console.WriteLine("Guest: {0}", isGuest);
```

Microsoft .net

Демонстрация #8

- Извличане на текущия Windows потребител и информация за него



Създаване на GenericPrincipal

- Стъпки при реализация на собствена автентикация и авторизация:
  1. Автентикиране на потребителя

```
if (ValidLogin(user, pass))
{ // User authenticated }
```

  2. Създаване на GenericIdentity и GenericPrincipal обекти

```
GenericIdentity id = new GenericIdentity("some user");
string[] roles = {"Manager", "Developer", "QA"};
GenericPrincipal prin = new GenericPrincipal(id,
roles);
```

  3. Закачане на Principal обекта за текущата нишка:

```
System.Threading.Thread.CurrentPrincipal = prin;
```

Авторизация по Principal

- Декларативна проверка:

```
[PrincipalPermission(SecurityAction.Demand,
Role="Developer", Authenticated=true)]
```

```
[PrincipalPermission(SecurityAction.Demand,
Name="Бай Киро")]
```

  - Ако текущата нишка не отговаря на посочения потребител или роля, се хвърля `SecurityException`
  - В имената на потребителите и ролите не се различават малки от главни букви

Microsoft .net

Авторизация по Principal

- Програмна проверка:

```
if (principal.IsInRole("Administrators"))
{
    // Perform some action
}
```

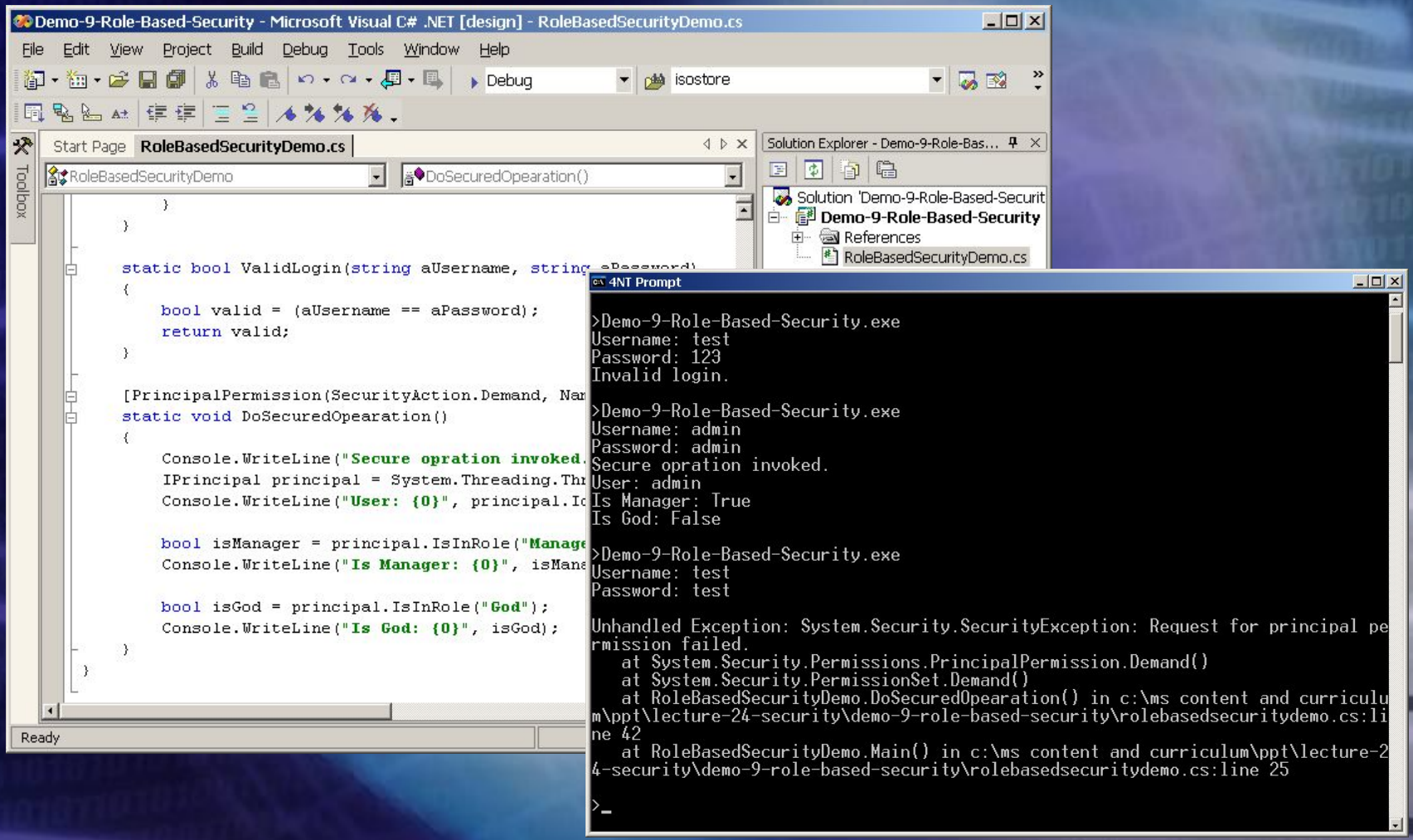
```
if (principal.Identity.Name == "Пешо")
{
    // Perform some action
}
```

```
PrincipalPermission prinPerm = new
    PrincipalPermission("Пешо", "Tester");
prinPerm.Demand();
// Throws SecurityException if the check fails
```

Microsoft .net

Демонстрация #9

- Авторизация с потребители и роли



Криптография в .NET Framework

- .NET Framework има силна поддръжка на криптографски алгоритми и технологии
- В `System.Security.Cryptography` са имплементирани:
  - Алгоритми за извличане на хеш:
    - MD5, SHA1, SHA256, SHA384, SHA512
  - Симетрични кодиращи алгоритми:
    - DES, 3DES, RC2, Rijndael/AES
  - Асиметрични кодиращи алгоритми:
    - RSA, DSA
  - Класове за работа с X.509 цифрови сертификати

Изчисляване на хеш стойност

- Премятане на SHA1 хеш стойност:

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

…

Console.Write("Enter some text: ");
string s = Console.ReadLine();
byte[] data = Encoding.ASCII.GetBytes(s);

SHA1CryptoServiceProvider sha1 =
    new SHA1CryptoServiceProvider();
byte[] sha1hash = sha1.ComputeHash(data);

Console.WriteLine("SHA1 Hash: {0}",
    BitConverter.ToString(sha1hash));
```

Microsoft .net

Подписване на XML (XMLDSIG)

- Подписването на XML документи става по стандарта на W3C
  - "XML-Signature Syntax and Processing"
  - http://www.w3.org/TR/xmldsig-core/
- В .NET Framework има пълна имплементация на стандарта в пакета `System.Security.Cryptography.Xml`
  - Подписване и проверка на подпис
- Основни класове:
  - `SignedXml` – осигурява изготвяне и проверка на XML сигнатури
  - `DataObject` – съдържа данните, които ще бъдат подписани

Microsoft .net

Сигурност в .NET Framework

Въпроси?

Упражнения

1. Опишете ключовите характеристики на сигурността в .NET Framework – безопасност на типовете, защита на паметта, защита от аритметични грешки, подписване на асемблитата, IsolatedStorage, Code Access Security, Role Based Security и др.
2. Напишете библиотека (Class Library проект във VS.NET), която съдържа клас със статичен метод `PrintVersion()`, който отпечатва на конзолата версията на асемблито, от което е зареден класа. Компилирайте асемблито в 2 различни версии (1.0 и 2.0), подпишете ги, направете ги със силни имена и ги инсталирайте в GAC. Реализирайте 2 конзолни приложения, които ползват съответно версия 1.0 и 2.0 на асемблито.

Упражнения

3. Напишете Windows Forms контрола за IE, която позволява създаване на албуми със снимки, които се съхраняват в IsolatedStorage за текущия потребител. Контролата трябва да позволява разглеждане на албума, добавяне и изтриване на снимки, които се съхраняват в IsolatedStorage.
4. Създайте Windows Forms контрола за IE, която може да отваря, редактира и записва текстови файлове на локалния диск на потребителя. По подразбиране отварянето на локален файл няма да работи. Направете асемблито на контролата да има силно име. Чрез Security Policy Editor дайте права за четене и писане на асемблито на контролата, като създадете Code Group по силното му име.

Упражнения

5. Напишете Windows Forms приложение, което позволява създаване и записване на текстови бележки. Приложението трябва да съхранява бележките във файл в профила на текущия потребител, ако има права за това или в IsolatedStorage ако няма права. Правата трябва да се проверяват програмно.
6. Напишете библиотека (DLL), която поддържа функционалност за регистриране на потребител по username и password и проверка на валидността на двойка username/password. Библиотеката трябва съхранява данните си в XML файл и да използва собствените си права за достъп до файла. Клиенти с ниски права, които не могат да четат файла, трябва да могат да ползват функционалността на библиотеката.

Упражнения

7. С помощта на Role Based Security направете приложение, което управлява потребителите в дадена система. Потребителите, техните пароли и ролите на всеки потребител трябва да се съхраняват в XML файл. Възможните роли за всеки потребител са Guest, User и Admin. Гостите в системата имат право да се регистрират и нищо друго. Потребителите в системата имат право да извличат списъка от всички регистрирани потребители. Администраторите имат право да редактират данните и ролите на всички потребители. При начално стартиране системата трябва да предлага форма за автентикация, която позволява влизане като някакъв потребител или влизане като гост без парола. Проверката на ролите да се реализира чрез `GenericPrincipal`.

Упражнения

8. Реализирайте приложението от предходната задача, като съхранявате паролите на потребителите не като чист текст, а като SHA1 хеш стойност. Дава ли това по-голяма сигурност за системата?

Microsoft .net

Използвана литература

- Svetlin Nakov, Implementing Application Security Using the Microsoft .NET Framework – Lecture at the National Conference "Information Technologies in the Education – A Necessary Investment for the Future of Bulgaria", Sofia, April 2004
- MSDN Lectures, Implementing Application Security Using the Microsoft .NET Framework – http://downloads.microsoft.co.za/MSDNEssentials/20040402/AppSecurity.ppt
- Understanding .NET Code Access Security – http://www.thecodeproject.com/dotnet/UB_CAS_NET.asp
- MSDN Library – http://msdn.microsoft.com

Българска асоциация на разработчиците на софтуер

НАПС
НАЦИОНАЛНА АКАДЕМИЯ ПО РАЗРАБОТКА НА СОФТУЕР