2.NET – Framework Fundamentals

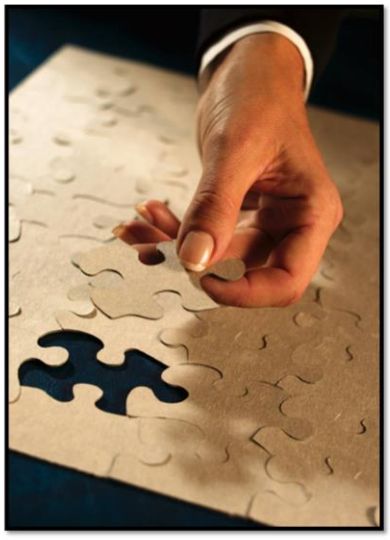
Academia Microsoft

Modulul 8

Securizarea execuției codului și a resurselor

Overview

- Folosirea **Code Access Security**
- Personalizarea modului de autentificare și autorizare



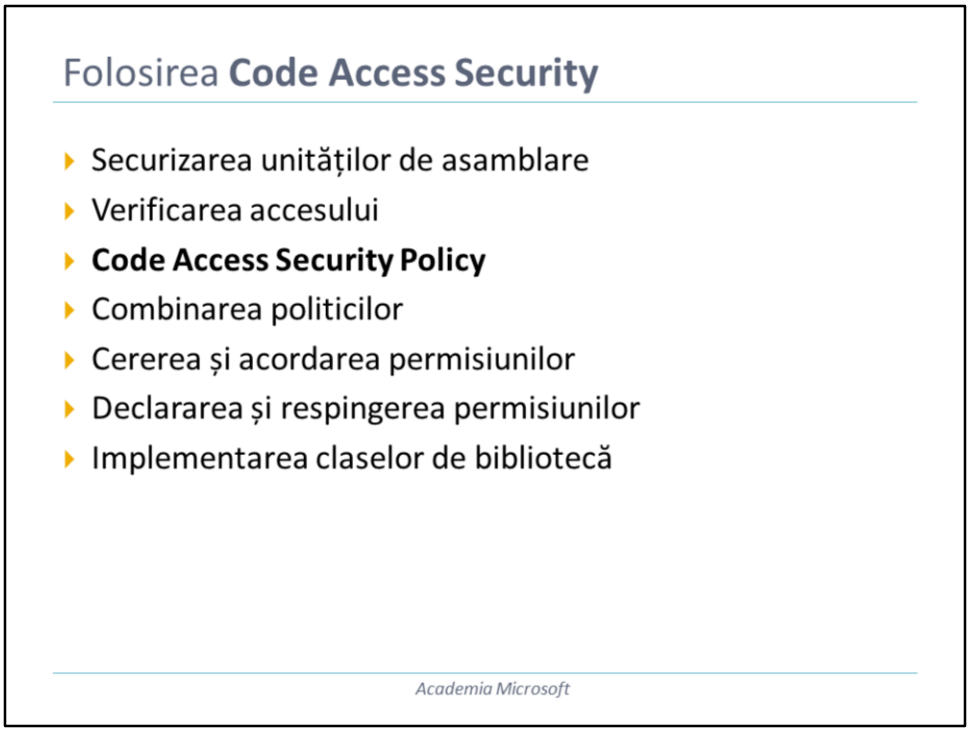
Academia Microsoft

Tot codul **Microsoft .NET Framework** este supus verificării accesului de .**NET Framework Code Access Security** (**CAS**) și a sistemului de operare Windows. Acest modul explică modul de funcționare **CAS**, cum puteți realiza verificări de permisiuni CAS în codul dumneavoastră și cum să configurați politica de securitate **CAS**. De asemenea acest modul descrie verificările de acces Windows și modul de folosire al bibliotecii .**NET Framework** pentru a citi și modifica listele de control al accesului Windows (**ACL-**uri).

Obiective

După completarea acestui modul veţi fi capabili să:

- Descrieți și implementați **CAS** folosind namespace-urile **System.Security** și **System.Security.Permissions**.
- Folosiți namespace-ul **System.Security.Principal** pentru a autentifica și autoriza utilizatori



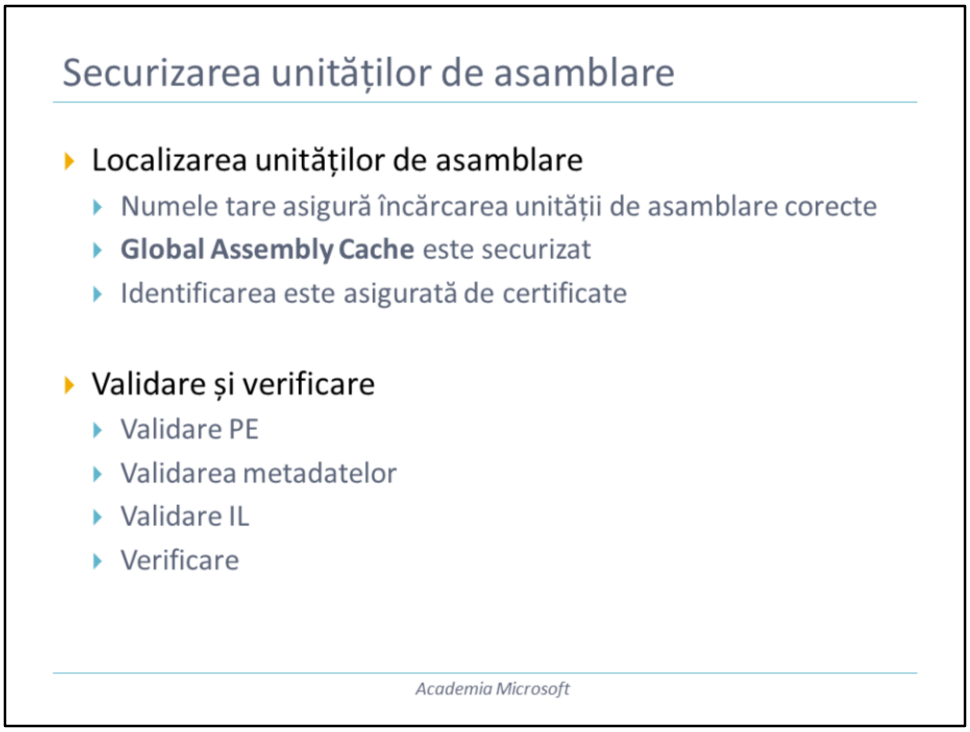
Securitatea Windows este bazată pe verificarea accesului utilizatorilor. Tot codul .**NET Framework** rulează sub modelul de securitatea Windows. Codul mobil rulează folosind privilegiile de securitate ale utilizatorului curent, ceea ce este o potențială amenințare de securitate.

.**NET Framework** introduce un nou nivel de securitate unde permisiunile sunt acordate codului în funcție de unitatea de asamblare. .**NET Framework CAS** nu poate acorda mai multe privilegii decât cele acordate de securitatea Windows, iar **CAS** de obicei acordă și mai puține privilegii. **CAS** folosește diferite tipuri de dovezi, iar dovezile cel mai des folosite fac referire la originea unității de asamblare.

Obiective

După completarea acestui modul veţi fi capabili să:

- Explicați elementele de bază **CAS**.
- Explicați și folosiți namespace-ul **System.Security.Policy**
- Folosiți clasele din namespace-ul **System.Security** și **System.Security.Permissions**



.**NET Framework** realizează multe verificări asupra unităților de asamblare, înainte de rularea acestora. Runtime realizează verificări asupra originii unităților de asamblare și asigură faptul că acestea conțin cod valid.

Încărcarea unei unități de asamblare

O unitate de asamblare .**NET Framework** este un fișier executabil portabil Windows (**PE**). Un fișier **PE** negestionat are un punct de intrare care este apelat de Windows.

Acesta este un potențial risc de securitate deoarece un virus se poate atașa executabilului și schimba punctului de intrare, astfel încât să ruleze codul virusului.

.**NET Framework** previne această amenințare deoarece codul .NET Framework nu este rulat de la punctul de intrare.

Runtime realizează multe verificări de securitate înainte de rularea oricărui cod. Aceste verificări sunt realizate pentru a asigura că nu există probleme în metadatele care descriu tipurile din unitatea de asamblare sau limbajul intermediar (**IL**) care este folosit pentru a implementa tipurile.

Aceste verificări sunt enumerate în tabelul următor:

Verificarea securității	Descriere
Validare PE	.NET Framework verifică informațiile care descriu fișierul PE, să nu fie corupte, și că face referire la locații valide în fișier.
Validarea metadatelor	.NET Framework verifică tabelele de metadate astfel încât să conțină informații valide. Aceste tabele conțin informații despre tipuri, module și unități de asamblare folosite de unitatea de asamblare curentă.
Validare IL	.NET Framework verifică fiecare cale de cod din unitatea de asamblare și codul IL să fie valid.
Verificare	.NET Framework încearcă să verifice că rularea codului nu face nimic nesigur. Această verificare se realizează la compilarea JIT. Codul care este verificat poate fi rulat atât timp cât unitatea de asamblare are suficiente permisiuni.

Dacă oricare dintre etapele de validare eșuează, codul unității de asamblare nu va fi rulat. Dacă testul de verificare eșuează atunci unitatea de asamblare va fi tratată ca și cum conține cod *neverificabil.* Există o permisiune **CAS** care vă permite să rulați cod neverificabil, dar această permisiune este acordată doar unităților de asamblare de încredere.

Puteți verifica dacă o unitate de asamblare este verificabilă folosind utilitarul în linie de comandă **peverify.exe**.

Localizarea unităților de asamblare

Un posibil risc de securitate poate fi atunci când cod străin poate forța unitatea de asamblare să ruleze respectivul cod, și nu biblioteca care trebuia inițial încărcată. .**NET Framework** previne acest lucru folosind o tehnologie numită **Fusion**.

Un dezvoltator .NET Framework poate da unei unități de asamblare un nume tare, adică un nume unic asociat unei anumite unități de asamblare. Acest nume nu mai poate fi acordat niciunei alte unități de asamblare.

Un nume tare implică semnarea unității de asamblare, adică un hash este calculat pentru unitatea de asamblare și este criptat folosind o cheie privată. Hash-ul și cheia publică sunt puse în fișierul **PE**, astfel încât atunci când o unitate de asamblare trebuie să folosească biblioteca cu nume tari, numele tare este scris în tabelele cu metadate ale unității de asamblare apelante.

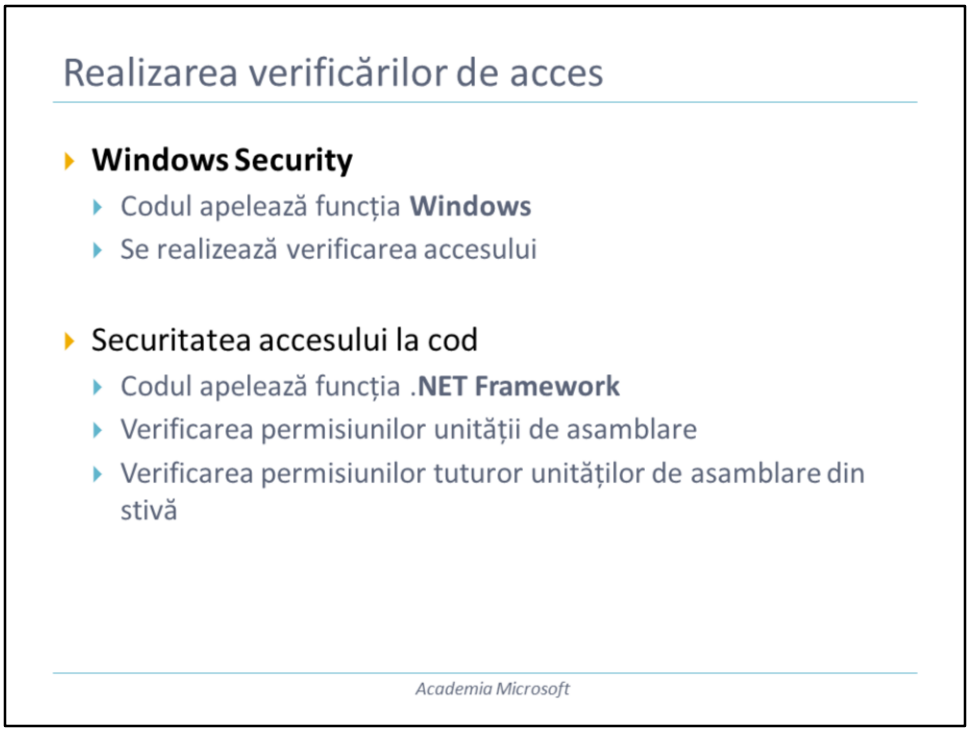
Când este cerută biblioteca la runtime, **Fusion** localizează unitatea de asamblare și verifică că aceasta este unitatea de asamblare corectă. Pentru a realiza acest lucru, **Fusion** calculează hash-ul pentru unitate, extrage hash-ul semnat din unitatea de asamblare și îl decriptează folosind cheia publică. Dacă cele două hash-uri corespund, unitatea de asamblare este încărcată.

Hash-ul și numele tare al unității de asamblare pot fi folosite ca dovezi în **CAS**.

Authenticode

Semnarea unei unități de asamblare nu garantează identitatea publicantului. Dacă aveți un certificat X.509, îl puteți folosi pentru a semna unitatea de asamblare folosind **Microsoft Authenticode**.

.**NET Framework** va detecta faptul că unitatea de asamblare este semnată folosind un certificat și furnizează aceste informații ca dovezi adiționale pentru **CAS**. Un certificat valid oferă identificare pozitivă a publicantului unității de asamblare.



CAS este bazat pe unități de asamblare, ceea ce înseamnă că unitații de asamblare îi sunt acordate permisiuni. De asemenea este bazat pe stivă, ceea ce înseamnă că atunci când o metodă rulează cod care necesită acordarea unei permisiuni, .**NET Framework** va verifica fiecare metodă din stivă, pentru a asigura că unitățile de asamblare au permisiuni. Dacă o unitate de asamblare din stivă nu are permisiunea, codul protejat nu este rulat. Acest mecanism asigură că dacă o unitate de asamblare cu permisiuni suficiente care apelează o unitate de asamblare cu mai puține permisiuni, codul de joasă încredere nu va avea permisiunile unității de asamblare apelante.

Thread-Based Access Token Security

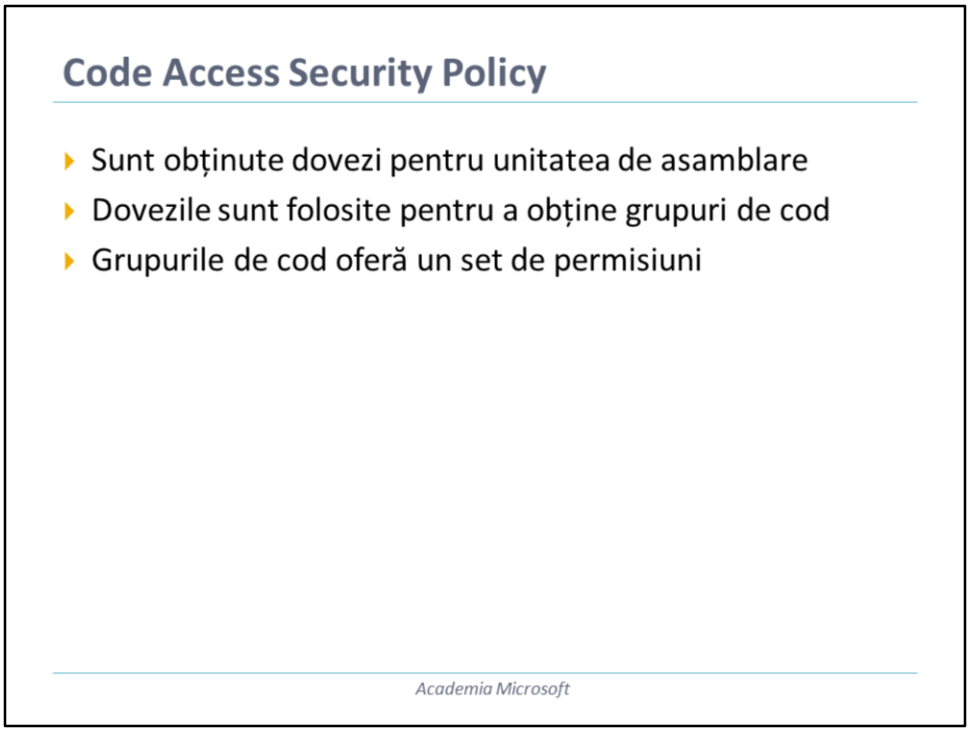
Securitatea **Windows** este bazată pe **thread-uri**. Windows creează un proces cu un **token** de acces la securitate, și implicit fiecare thread din proces va avea acest token de acces. Un thread Windows poate lua locul unui utilizator, dacă are credențialele corecte de logare; când acest lucru se întâmplă, thread-ul va avea **token-ul** de acces cu privilegiile respectivului utilizator.

Când un thread folosește obiecte de securizare Windows (fișiere, obiecte de sincronizare), Windows realizează o verificare automată a accesului pentru a vedea dacă thread-ul are permisiuni de a folosi respectivul obiect de securizare. Orice cod apelat de thread-ul current va deține **token-ul** de acces, deoarece **token-ul** de acces este bazat pe thread-uri; astfel dacă o aplicație negestionată apelează cod de bibliotecă, acest cod va avea aceleași permisiuni ca ale thread-ului.

Cu toate că multe din clasele .**NET Framework** apelează obiecte de securizare Windows și sunt subiectul unor verificări de **token** de acces. Acest cod este protejat de **CAS**, astfel beneficiați de ambele mecanisme de securitate. .**NET Framework** furnizează două tipuri de de securitate bazată pe thread-uri: securitate bazată pe roluri .**NET Framework Principal** și securitate bazată pe roluri **Enterprice Services**.

Assembly-Based Code Access Security

CAS depinde de unităţile de asamblare. Când codul apelează o metodă protected, .**NET Framework** verifică unitatea de asamblare pentru permisiunile necesare metodei, apoi parcurge stiva pentru a verifica fiecare unitate de asamblare de aceste permisiuni. Dacă o unitate de asamblare nu are toate permisiunile necesare este ridicată o excepție de securitate și codul nu va fi rulat.



.**NET Framework CAS** este foarte flexibil și configurabil. Puteți cofigura politica de securitate pentru un grup de computere, un computer sau un utilizator individual, iar aceste grupuri sunt combinate când este realizată o verificare a permisiunilor.

Termenii politicii de securitate

Fiecare unitate de asamblare are dovezi asociate care identifică originea unității de asamblare; aceste dovezi sunt adunate la încărcarea unității de asamblare și sunt folosite la runtime ca intrare pentru politica **CAS**. Politica poate fi definită pentru întreprinderi, computer sau un utilizator particular. Fiecare politică **CAS** conține grupuri de cod și fiecare grup de cod definește seturi de permisiuni pentru o anumită dovadă. Un set de permisiuni reprezintă una sau mai multe permisiuni. Codul poate folosi **CAS** prin cererea ca o permisiune să fie acordată. Acești termeni sunt descriși în tabelul următor:

Termen	Descriere
Permisiune	Sunt acordate una sau mai multe permisiuni unităților de asamblare. Codul poate cere ca fiecare unitate de asamblare din stivă să dețină una sau mai multe permisiuni.
Set de permisiuni	O colecție de una sau mai multe permisiuni.

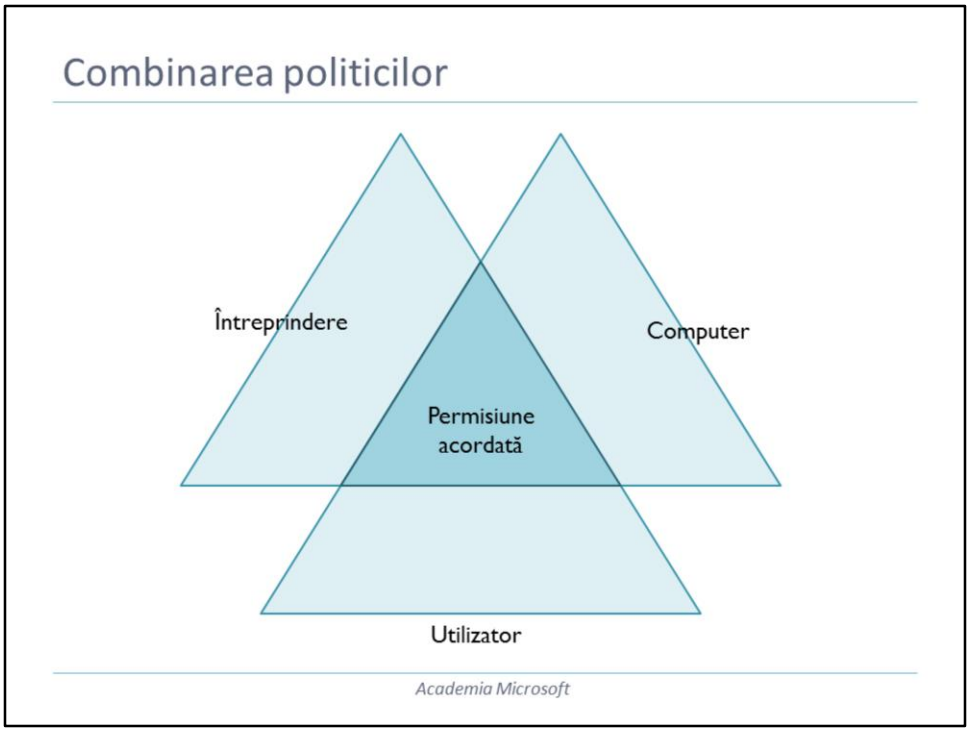
Termen	Descriere
Grup de cod	Un grup de cod definește setul de permisiuni care va fi acordat pentru o unitate de asamblare cu o anumită valoare a dovezii. Spre exemplu, un grup de cod poate indica că o unitate de asamblare descărcată de pe un computer pe rețeaua locală poate avea acces la fișiere dintr-un anumit folder.
Dovezi	Reprezintă proprietatea unei unități de asamblare, precum certificarea X.509 pentru o unitate de asamblare semnată. CAS folosește dovezile și grupuri de cod pentru a determina setul de permisiuni acordate unei unități de asamblare.
Politică	Fiecare obiect politică determină securitatea definită pentru întreprindere, computer, utilizator sau domeniul de aplicație curent. Fiecare politică va genera o colecție de permisiuni, iar setul final de permisiuni este o combinație din permisiunile acordate de fiecare politică.

Grupurile de cod acordă permisiuni; ele nu refuză permisiuni. Dacă doriți ca o unitate de asamblare să nu aibe o anumită permisiune, trebuie să asigurați un grup de cod ca dovadă, care să identifice unitatea de asamblare că nu are respectiva permisiune. Fiecare nivel de politică va genera un set de permisiuni, iar ultimul set de permisiuni acordat unității de asamblare este combinat.

Cereri de permisiuni

Există două modalități de a realiza o cerere: imperativă sau declarativă. Pentru a realiza o cerere imperativă, creați un obiect permisiune, adică o instanță a unei clase care implementează interfața **IPermission**, apoi apelați metoda **IPermission.Demand**. Această metodă declanșează o parcurgere a stivei la runtime, unde unitatea de asamblare pentru fiecare metodă din stivă este verificată pentru respectiva permisiune. Dacă o unitate nu are această permisiunea, metoda **Demand** va arunca o excepție de tipul **SecurityException**.

Pentru a realiza o cerere declarativă, marcați o metodă cu un atribut derivat din clasa **SecurityAttribute**. Acest tip de securitate prezintă avantajul că informațiile de securitate sunt stocate ca metadate în unitatea de asamblare, care pot fi citite de utilitare externe.



Politicile folosesc dovezile pentru a determina setul de permisiuni pentru o unitate de asamblare. Există patru tipuri de politici. Trei politici – **Enterprise**, **Machine** și **User** – sunt încărcate automat din fișiere XML. Administratorii de sistem și utilizatorii pot schimba politica prin folosirea utilitarului **caspol** sau **mscorcfg.msc** din **SDK-ul .NET Framework**. A patra politică, **AppDomain**, este furnizată programatic, când este creat domeniul de aplicație.

Politică de investigare

Sunt patru niveluri de politică definite de enumerația **PolicyLevelType**: **Enterprise**, **Machine**, **User** și **AppDomain**. Valorile pentru politica **AppDomain** sunt furnizate de codul care creează domeniul de aplicație. Pentru celelalte trei politici, aceste informații sunt stocate în fișiere XML. În următorul tabel sunt listate locația acestor fișiere.

Politică	Locație
User	%appdata%\Microsoft\CLR Security Config\%FrameworkVersion%\security.config
Machine	%windir%\Microsoft.NET\Framework\%FrameworkVersion%\config\security.config
Enterprise	%windir%\Microsoft.NET\Framework\%FrameworkVersion%\config\enterprisesec.config

%appdata% oferă folderul de date al aplicației pentru utilizatorul curent, %windir% oferă directorul Windows, iar %FrameworkVersion% oferă versiunea .NET Framework.

Aceste fișiere sunt stocate pe hard disk și sunt subiectul securității sistemului de fișiere **NTFS**; acest lucru înseamnă că un utilizator își poate modifica fișierul de politică, dar doar administratorii pot schimba fișierele de politică **Machine** și **Enterprise**. Totuși, toate aceste fișiere pot fi citite de orice utilizator, adică aplicațiile .NET Framework pot folosi aceste politici la runtime.

Permisiunile declarative

Unitățile de asamblare pot avea permisiuni declarative, caz în care toate tipurile din unitatea de asamblare sunt afectate. În următorul tabel sunt prezentate valorile **SecurityAction** pe care le puteți aplica.

SecurityAction	Descriere
RequestMinimum	Indică minimul de permisiuni necesare tipurilor din unitatea de asamblare.
RequestRefuse	Indică faptul că toate tipurile din unitatea de asamblare trebuie să ruleze fără respectiva permisiune.
RequestOptional	Anumite tipuri din unitatea de asamblare pot cere o permisiune.

Puteți folosi orice atribut de permisiune cu aceste valori **SecurityAction**.

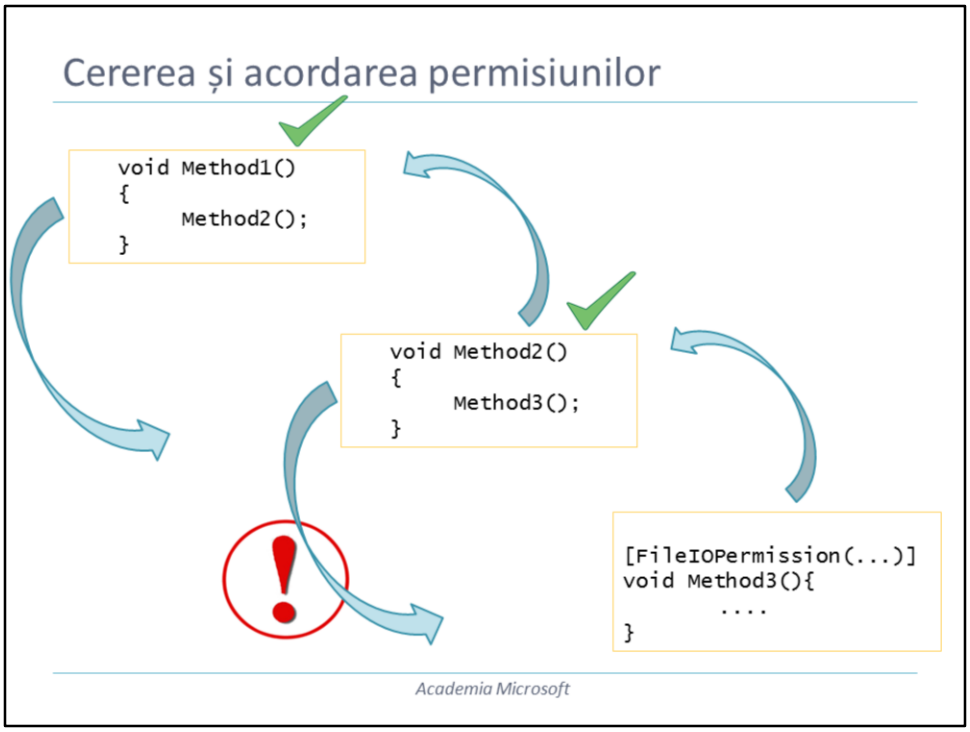
Acordarea seturilor de permisiuni folosind politica de combinare

Când este realizată o cerere, la runtime sunt determinate permisiunile unității de asamblare. Pentru acest lucru, sunt parcurse următoarele etape:

- Runtime folosește fiecare politică pentru a determina setul de permisiuni pentru unitatea de asamblare. **CAS** folosește intersecția seturilor de permisiuni acordate de fiecare politică. Adică unitatea de asamblare obține permisiunile comune tuturor seturilor de permisiuni.

- Runtime calculează uniunea seturilor de permisiuni minime (**RequestMinimum**) și opționale (**RequestOptional**), specificate de unitatea de asamblare, și determină intersecția cu setul de permisiuni acordat de politică.

- Runtime înlătură orice permisiune specificată de unitatea de asamblare (setul de permisiuni **RequestRefuse**).



Puteți proteja codul folosind testele de permisiuni **CAS**. Aceste teste pot fi fie imperative (prin folosirea unei cereri pe un obiect permisiune), fie declarative (prin folosirea unui atribut permisiune pe o clasă sau metodă). Atunci când este apelat cod care necesită o permisiune, la runtime se realizează o parcurgere a stivei și se verifică teste asupra permisiunilor tuturor unităților de asamblare din stivă. Codul rulează doar dacă toate unitățile de asamblare au permisiunea necesară.

Realizarea testelor de acces

Când apelați o metodă folosind metoda **Assert** se întâmplă următoarele:

- Runtime apelează metoda.
- **CAS** parcurge stiva, pentru a verifica că unitățile de asamblare din aceasta au permisiunea necesară.
- Parcurgerea stivei se oprește când este apelată metoda **Assert**.

Interfața IPermission și clasa PermissionSet

Scopul clasei **PermissionSet** este de a conține unul sau mai multe obiecte care implementează **IPermission**, astfel acțiunile realizate asupra setului de permisiuni sunt realizate asupra tuturor obiectelor permisiune conținute.

Codul apelează metoda **Demand** pe o clasă care implementează interfața **IPermission** pentru a indica realizarea unei acțiuni privilegiate, doar de codul cu o permisiune specifică.

Apelarea metodei **Demand** implică verificarea fiecărei metode din stivă începând cu metoda apelantă. Este verificată acordarea permisiunii unității de asamblare care conține tipul sau metoda.

Obiectele permisiune și clasa **PermissionSet** au două metode: **FromXML** și **ToXML**, aceste metode permit seturilor de permisiuni să fie create și exportate din XML-uri. Acesta reprezintă modul de realizare a obiectelor **PermissionSet** .NET Framework.

Cererea permisiunilor

Constructorul unui atribut permisiune derivat din clasa **CodeAccessSecurityAttribute** primește un parametru de tipul **SecurityAction**, pentru a indica tipul de acțiune de securitate necesară la runtime. Pentru a indica o parcurgere a stivei trebuie folosită valoarea **SecurityAction.Demand**. Fiecare atribut de securitate are proprietăți adiționale care specifică informații despre permisiunea cerută. Spre exemplu, o metodă care va cere acces read-only la un fișier din folderul C:\MyFolder poate realiza următoarea cerere:

```
[FileIOPermission ( SecurityAction.Demand, Read = @"C:\MyFolder")]
void AccessFiles()
{
      FileStream fs = File.OpenRead(@"C:\MyFolder\MyFile.txt");
      ...
}
void CallMethod()
{
      try
      {
            AccessFiles();
      }
      catch (SecurityException se)
      {
            Console.WriteLine("The method {0} does not have the
                  permission {1}", se.Method.ToString(),
                        se.Demanded.ToString());
      }
}
```

Dacă apelantul metodei **AccessFiles** nu are această permisiune atunci este aruncată o excepție **SecurityException**. Codul dumneavoastră poate accesa proprietăți pe acest obiect pentru a determina setul de permisiuni și permisiunea cerută.

Atenție! Constructorul clasei **FileStream** va realiza propriile cereri de permisiuni. Metoda **File.OpenRead** apelează constructorul **FileStream**.

Avantajul cererilor declarative este ca acel cod să folosească reflecția pentru a determina cererile necesare codului dumneavoastră. Utilitarul **permcalc.exe** va parcurge fiecare cale de cod dintr-o unitate de asamblare și va determina setul minim de permisiuni necesar unității de asamblare astfel încât să ruleze fără a arunca o excepție de tipul **SecurityException**.

Pentru o abordare mai fină puteți folosi cereri imperative. Pentru acest lucru, creați o instanță a unei clase care implementează interfața **IPermission** sau un obiect **PermissionSet**. Puteți apela metoda **Demand** pentru a iniția o parcurgere a stivei.

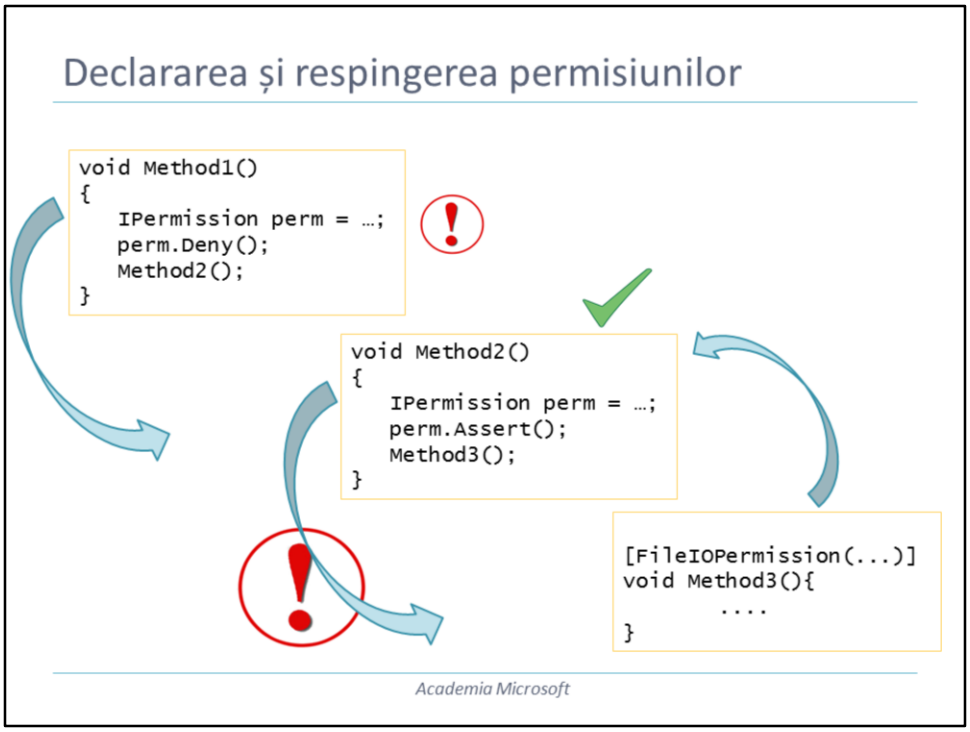
Avantajul cererilor imperative este că le puteți folosi pentru a proteja doar o parte a unei metode. Spre deosebire, la abordarea declarativă, catitatea minimă de cod pe care o puteți modifica este o întreagă metodă. În plus, codul poate realiza cereri bazate pe informații obținute la runtime. Următorul cod prezintă modul de implementare a metodei **AccessFiles** prin folosirea cererilor imperative:

```
void AccessFiles()
{
      FileIOPermission perm = new FileIOPermission(
            FileIOPermissionAccess.Read, @"C:\Myfolder");
      perm.Demand();
      FileStream fs = File.OpenRead(@"C:\MyFolder\MyFile.txt");
      ...
}
```

Puteți aplica unul sau mai multe atribute permisiune pe o metodă sau clasă, caz în care ambele permisiuni trebuiesc să fie acordate unităților de asamblare apelante.

Clasele pentru permisiunea accesului la cod sunt derivate din clasa **CodeAccessPermission**, clasă care implementează interfața **IPermission** și interfața **IStackWalk**. Pentru fiecare clasă pentru permisiunea accesului la cod există o clasă atribut derivată din clasa **CodeAccessSecurityAttribute**. În tabelul următor sunt descrise cele mai comune clase pentru permisiune **CAS**.

Clasă	Descriere
FileIOPermission	Codul apelant trebuie să aibe permisiunile de citire și scriere la accesarea fișierelor.
RegistryPermission	Codul apelant trebuie să aibe permisiunile de citire, scriere și creare la accesarea fișierelor.
UIPermission	Codul apelant trebuie să aibe permisiunile necesare pentru accesarea clipboard-ului sau pentru afișarea căsuțelor de dialog.



Codul .NET Framework poate schimba comportamentul testelor pentru seturile de permisiuni pentru parcurgerea stivei, prin folosirea valorilor **SecurityAction** (sau metodele **IStackWalk: Deny**, **PermitOnly** și **Assert)**. Acești modificatori acționează numai asupra metodei curente sau până când o metodă potrivită **Revert** este apelată asupra clasei **CodeAccessPermission**.

Respingerea unei permisiuni

Ca dezvoltatorul unei componente, nu aveți control asupra codului care va apela respectiva componentă; astfel nu puteți garanta că practici precum privilegii minime vor fi folosite. Metoda **IStackWalk.Deny** indică faptul că orice cerere pentru o anumită permisiune va eșua de fiecare dată indiferent dacă unitatea de asamblare deține respectiva permisiune. Acest lucru permite codului dumneavoastră să ruleze folosind permisiuni minime și este foarte util dacă sunt apelate alte părți de cod. Următorul exemplu ilustrează modul de folosire a unui modificator pentru parcurgerea stivei:

```
Void MyMethod(SomeObject obj)
{
            FileIOPermission perm = new FileIOPermission(
            FileIOPermissionAccess.Write, @"C:\");
            perm.Deny();
            obj.CallSomeMethod();
            CodeAccessPermission.RevertDeny();
}
```

În codul anterior metoda **MyMethod** nu are informații despre metoda **CallSomeMethod**, astfel minimizează riscurile prin refuzarea dreptului de scriere discului C:. După ce este apelată metoda **RevertDeny**, metoda **MyMethod** are dreptul de scriere pe discul C.

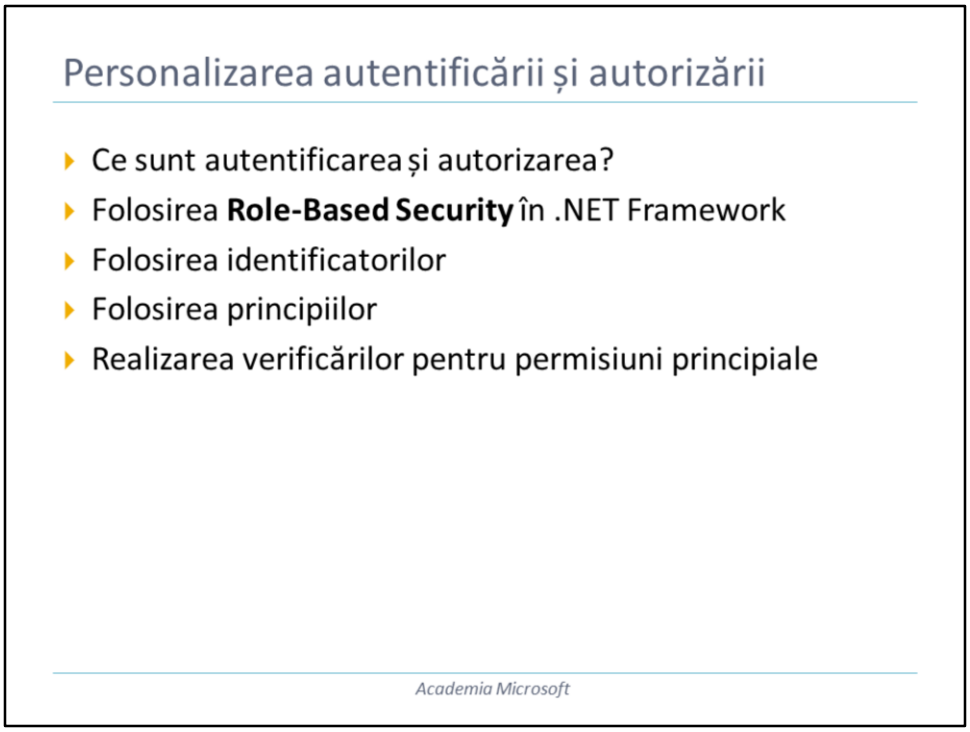
Specificarea unui set limitat de permisiuni

Metoda **Deny** este utilă pentru specificarea unui număr redus de permisiuni,dar pentru mai multe permisiuni apelați metoda **PermitOnly** (pentru un set de permisiuni). După apelarea metodei **PermitOnly**, runtime va genera o excepție, dacă oricare cod va cere o permisiune din afara setului de permisiuni acceptat. La fel precum metoda **Deny**, efectul metodei **PermitOnly** persistă până la ieșirea din metoda apelantă sau până când este apelată metoda **RevertPermitOnly**.

Declararea permisiunilor

Fiecare cerere cauzează o parcurgere a stivei, iar acesta poate reprezenta o operație costisitoare în ceea ce privește timpul și performanța. Pentru a minimiza acest cost runtime furnizează metoda **IStackWalk.Assert**.

Metoda **Assert** indică faptul că dacă este apelat codul de metoda curentă urmează o parcurgere a stivei pentru permisiunea specifică, care nu va depăși însă această metodă. Dacă unitatea de asamblare care apelează metoda **Assert**, nu deține această permisiune, apelarea metodei **Assert** nu va avea nici un efect.



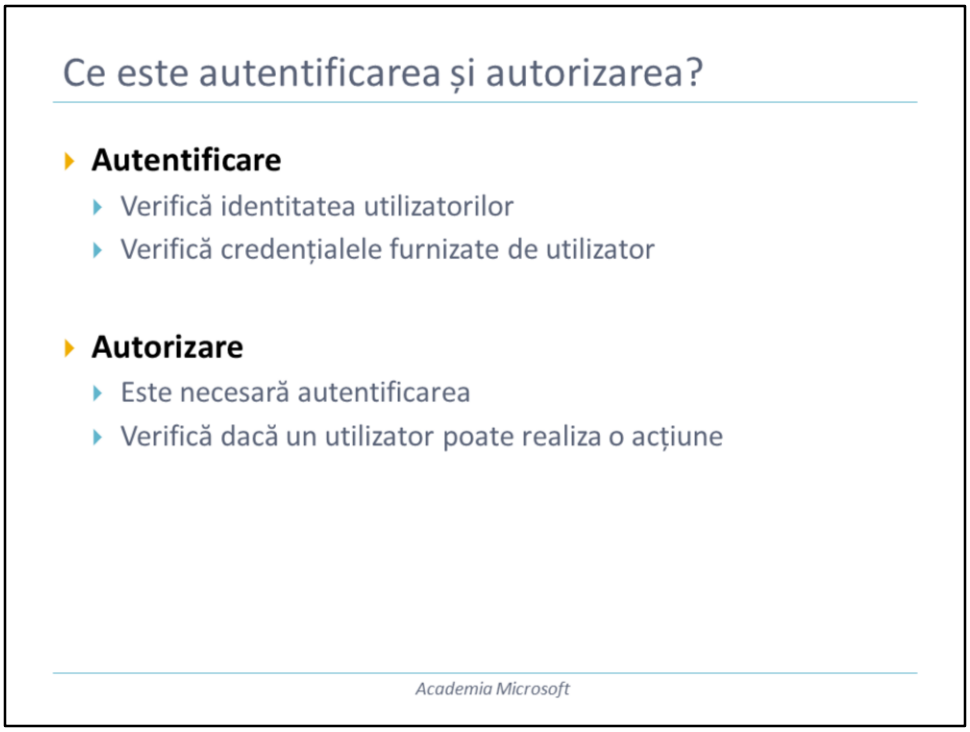
Autentificarea reprezintă un mecanism de verificare a utilizatorului și de validare a credențialelor acestuia. *.NET Framework* furnizează mecanisme de autentificare pentru aplicații ASP.NET, și vă permite să definiți propriile dumneavoastră scheme de utilizare și să setați principiul thread-ului la utilizatorul Windows autentificat.

Autorizarea reprezintă verificările de acces realizate, înainte ca un utilizator să poată apela o anumită metodă sau clasă. Un utilizator este reprezentat de un obiect principial care poate fi atașat la un *thread.* Puteți utiliza atribute sau obiecte permisiune pentru a cere ca principiul de thread curent, să fie un anumit utilizator, sau un utilizator cu un anumit rol.

Obiective

După completarea acestui capitol veți fi capabil să:

- Explicați conceptele de autentificare și autorizare.
- Explicați principiile securității bazate pe roluri în *.NET Framework.*
- Scrieți clase personalizate pentru autentificare și autorizare folosind namespace-ul **System.Security.Principal**.



Autentificarea este un mecanism pe care îl folosiți pentru a vă asigura de identitatea unui utilizator. Sunt multe modalități de autentificare; unele sunt simple, precum o verificare de parolă, iar alte scheme pot fi foarte complexe, precum o scanare a irisului sau a amprentei digitale. Windows furnizează propria schemă de autentificare, astfel puteți folosi principiile generate de această schemă, sau să definiți propria dumneavoastră schemă de autentificare.

Autentificare

Cea mai potrivită schemă de autentificare este autentificarea Windows, deoarece nu trebuie să scrieți cod suplimentar. De asemenea, biblioteca *.NET Framework* oferă **Passport Authentification** și **Forms Authentification** pentru utilizatorii ASP.NET. Aspectul principal al acestor scheme este un mecanism de obținere a credențialelor (numele de utilizator și parola), autentificarea acestor credențiale, iar apoi furnizarea unei autentificări solide.

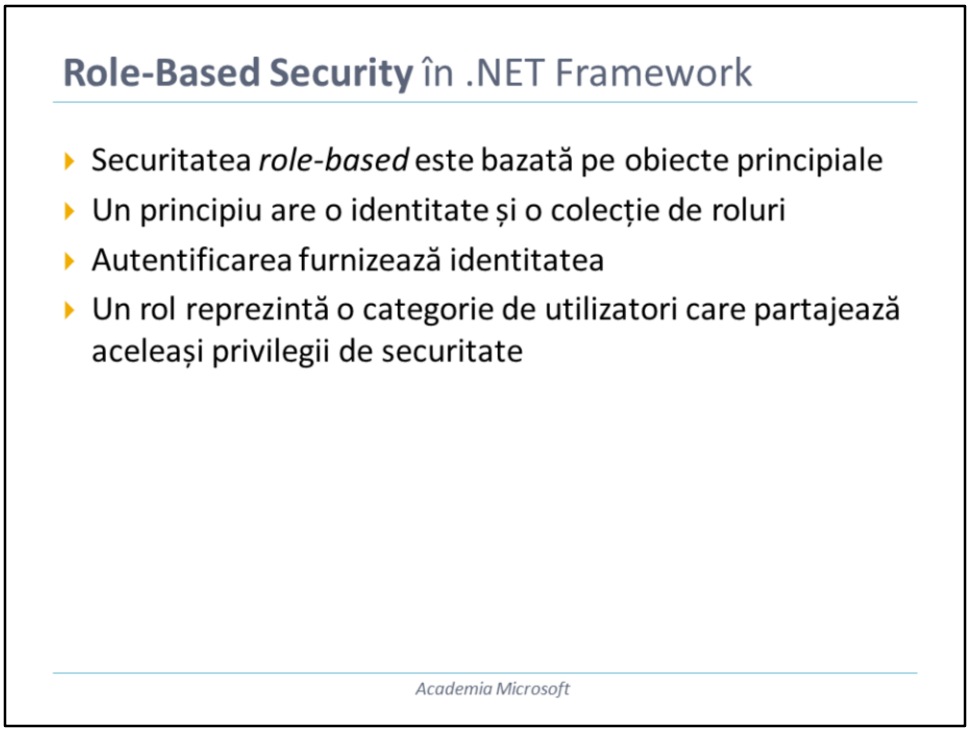
De asemenea, puteți să implementați propria dumneavoastră schemă de autentificare, folosind clase din biblioteca *.NET Framework*. Bibliotecile criptografice *.NET Framework* furnizează clase care vă permit să stocați informații și să securizați memoria folosită de codul pentru autentificarea credențialelor.

Autorizarea

Autorizarea este un mecanism pe care îl folosiți pentru a oferi utilizatorilor autentificați, autoritatea de a realiza o acțiune.

Autorizarea nu are sens fără autentificare. O schemă de autorizare poate permite utilizatorilor neautentificați să realizeze anumite acțiuni, însă acțiunea va avea privilegii minime, deoarece nu este cunoscută identitatea utilizatorului.

O schemă de autorizare implică realizarea unei verificări a accesului asupra unui token autentificat, care reprezintă utilizatorul. .*NET Framework* furnizează două mecanisme de securitate bazate pe roluri: roluri Enterprise Services și roluri bazate pe principii .*NET Framework*. Aceste două mecanisme sunt separate și au propriile beneficii și probleme. Rolurile **Enterprise Services** folosesc autentificarea Windows și sunt grupuri Windows. Dacă dezvoltați o aplicație **Enterprise Services**, atunci aceasta este cea mai bună opțiune.



Securitatea bazată pe roluri implică verificări de acces realizate prin folosirea de obiecte sau atribute permisiune. Acestea nu sunt permisiuni **CAS**, astfel cererea pentru o permisiune principială nu implică o parcurgere a stivei. În schimb, cererile de permisiune verifică obiectul principial atașat thread-ului curent, pentru a vedea dacă identitatea principiului a fost autorizată sau dacă principiul reprezintă un rol autorizat.

Securitatea bazată pe roluri

Securitatea bazată pe roluri implică realizarea de verificări de acces asupra obiectelor principiale. Aceste verificări folosesc arhitectura de permisiuni *.NET Framework*. La runtime sunt realizate verificări asupra atributelor sau obiectelor permisiune, însă, obiectele permisiune principiale nu sunt permisiuni **CAS**, și nu implementează interfața **IStackWalk**. Astfel o cerere pentru o permisiune principială nu va cauza o parcurgere a stivei.

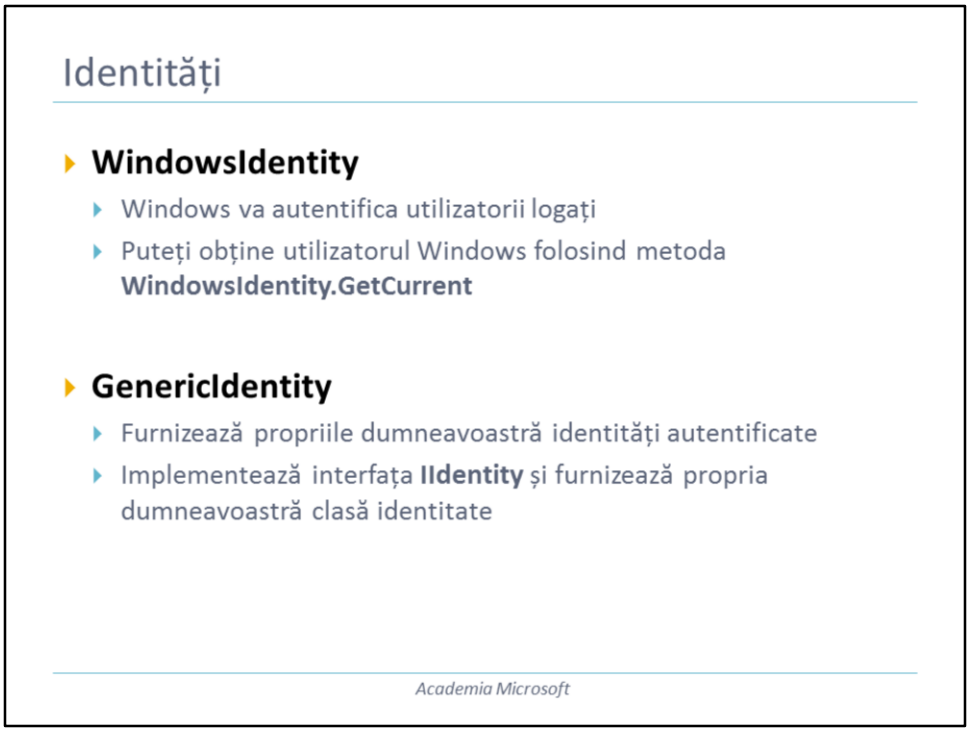
Un obiect principial este atașat unui thread, astfel principiul va fi întotdeauna disponibil, indiferent de codul rulat de thread.

Obiecte identitate

Mecanismul de autentificare furnizează identitatea utilizatorului, care este fie o identitate autentificată, fie neautentificată. Utilizatorii autentificați sunt vitali pentru mecanismul de autorizare pe care îl veți folosi. Biblioteca *.NET Framework* oferă două clase identitate, **GenericIdentity** și **WindowsIdentity**. Pentru a defini propria dumneavoastră clasă identitate implementați interfața **IIdentity**.

Obiecte principiale

Un obiect principial înglobează o identitate cu rolurile pe care utilizatorul le poate realiza. Biblioteca *.NET Framework* oferă două clase principiale: **GenericPrincipal** și **WindowsPrincipal**. Autorizarea bazată pe roluri este realizată asupra rolurilor din cadrul unui obiect principial. Pentru a scrie propria clasă principială (spre exemplu, dacă doriți să utilizați propria dumneavoastră schemă de autentificare pentru obiecte identitate), trebuie să implementați interfața **IPrincipal**.



Obiectele identitate sunt folosite pentru a identifica un anumit utilizator și sunt create de cod folosit pentru autentificarea credențialelor de utilizator. Dacă doriți să creați propria dumneavoastră schemă de autentificare, trebuie să creați și o clasă identitate, pentru a indica dacă utilizatorul este autentificat. Clasa **WindowsIdentity** este folosită pentru a obține informații despre utilizatorul Windows curent autentificat.

Clasa WindowsIdentity

Fiecare aplicație *.NET Framework* este subiectul autentificării Windows. Modul obișnuit de obținere a unei instanțe pentru această clasă este apelarea metodei **GetCurrent**. Această metodă returnează obiectul identitate Windows pentru thread-ul curent. Puteți folosi acest obiect pentru a lua decizii de autorizare, bazate pe numele de utilizator sau grupurile Windows pentru utilizator. Proprietatea **Name** a acestei clase va returna numele de utilizator Windows sub forma **UPN** (User Principal Name) : DOMAIN\USERNAME, unde DOMAIN reprezintă numele computerului curent pentru conturile locale, sau BUILTIN pentru conturile de sistem. Clasa mai prezintă o proprietate numită **Groups**, care este o colecție enumerabilă a grupurilor Windows pentru utilizator.

Proprietatea **User** a obiectului **WindowsIdentity** este un obiect de tipul **SecurityIdentifier**, astfel aveți acces la SID-ul utilizatorului.

În următorul exemplu, codul va afișa utilizatorul curent și grupurile Windows pentru acesta:

```
WindowsIdentity user = WindowsIdentity.GetCurrent();
Console.WriteLine("User:");
Console.WriteLine(user.Name);
Console.WriteLine("Groups:");
foreach( IdentityReference group in user.Groups )
{
     NTAccount ntGroup = group.Translate(typeof(NTAccount)) as
            NTAccount;
     Console.WriteLine(ntGroup.Value);
}
```

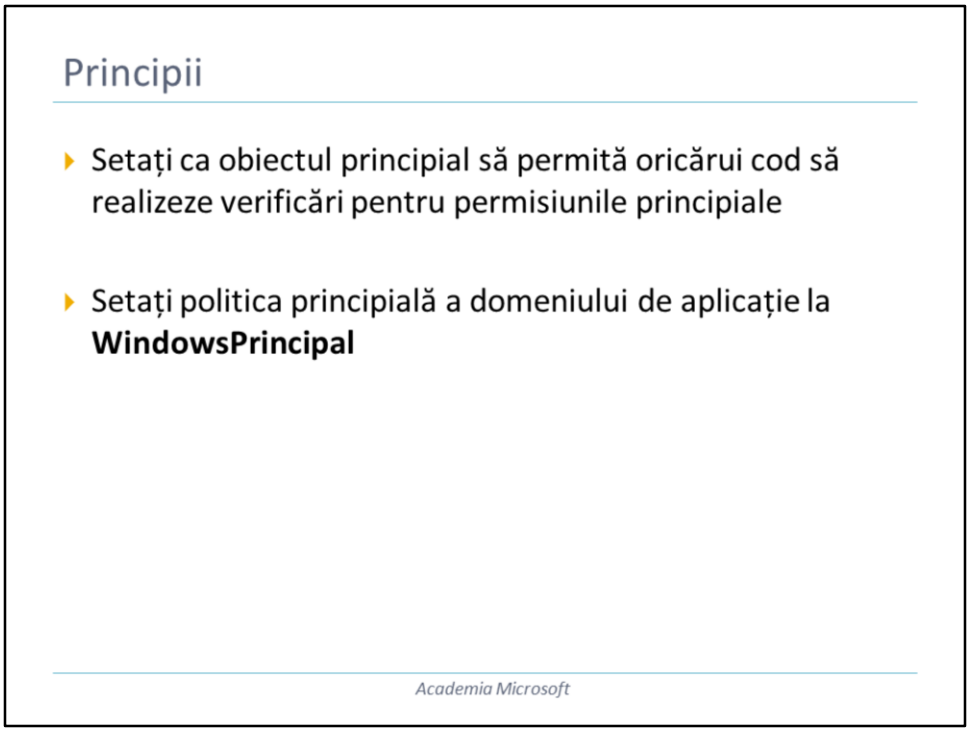
Obiecte identitate personalizate

Trebuie să creați propria clasă identitate dacă doriți să oferiți o scemă de autentificare. O clasă identitate personalizată trebuie să implementeze interfața **IIdentity**, care are trei proprietăți. În următorul tabel sunt descrise aceste proprietăți:

Proprietate	Descriere
AuthentificationType	Un string care descrie tipul de autentificare. Dacă oferiți o implementare proprie, puteți utiliza orice nume.
IsAuthenticated	O valoare booleană care determină dacă utilizatorul este autentificat. Aceasta este o proprietate importantă, folosită pentru a indica dacă utilizatorul a trecut verificările de autentificare.
Name	Un string ce reprezintă numele de utilizator. Numele este folosit de obiectele permisiune principiale la realizarea de verificări de autorizare pentru un utilizator.

Implementarea interfeței **IIdentity** este ușoară. Cel mai important task la crearea unei clase identitate personalizate este codul de autentificare, care autentifică credențialele și folosește rezultatele pentru a crea un obiect identitate.

Biblioteca *.NET Framework* furnizează o implementare simplă a unei clase identitate numită **GenericIdentity**. Obiectele acestei clase sunt create folosind numele utilizatorului și opțional numele schemei de autentificare. Obiectele acestei clase, indică întotdeauna că sunt autentificate.



Obiectele principiale înglobează obiecte identitate și oferă verificări bazate pe roluri. Fiecare thread are un principiu atașat, dar trebuie să setați politica principială pentru domeniul de aplicație să folosească un principiu mai puțin important.

Clasa WindowsPrincipal

Această clasă înglobează clasa **WindowsIdentity**, care realizează verificări de roluri asupra grupurilor Windows ale utilizatorului. Metoda **WindowsIdentity.GetCurrent** returnează obiectul **WindowsIdentity** pentru thread-ul curent.

Nu există o metodă echivalentă în clasa **WindowsPrincipal**; însă puteți da ca parametru un obiect **WindowsIdentity** pentru contructorul clasei **WindowsPrincipal**. Acest lucru este ilustrat în următorul exemplu.

```
WindowsPrincipal user = new WindowsPrincipal(
            WindowsIdentity.GetCurrent);
Console.WriteLine(”Is an administrator? {0}”,
            user.IsInRole(WindowsBuiltInRole.Administrator));
```

Clasa **WindowsPrincipal** prezintă trei versiuni ale metodei **IsInRole**, în plus față de metoda interfeței **IPrincipal**. Codul precedent prezintă modul în care această metodă verifică dacă utilizatorul este membru al unuia dintre grupurile standard Windows. Metoda primește ca parametru una dintre valorile enumerației **WindowsBuiltInRole**.

Crearea obiectelor principiale personalizate

O clasă principială personalizată trebuie să implementeze interfața IPrincipal, care are o metodă și o proprietate descrise în următorul tabel.

Membru	Descriere
IsInRole	Această metodă returnează o valoare booleană, pentru a indica dacă obiectul principial este pentru un utilizator cu rolul dat ca parametru metodei.
Identity	Această proprietate este un obiect identitate care implementează interfața IIdentity.

Metoda **IsInRole** primește ca parametru numele unui rol, și returnează **true,** dacă principiul reprezintă un utilizator cu rolul specificat.

Biblioteca *.NET Framework* furnizează o implementare simplă numită **GenericPrincipal**. Obiectele acestei clase sunt create folosind un obiect identitate și un vector de string-uri pentru roluri. Clasa implementează metoda **IsInRole**. Implementarea returnează dacă rolul specificat este în vectorul de string-uri dat ca parametru constructorului.

Principii și thread-uri

Fiecare thread are un principiu atașat pe care îl puteți obține folosind proprietatea **Thread.CurrentPrincipal**. Veți folosi acest principiu pentru verificări de permisiuni principiale bazate pe roluri, și puteți seta principiul de thread pentru a personaliza obiectul principial, deoarece proprietatea poate fi atât citită cât și scrisă.

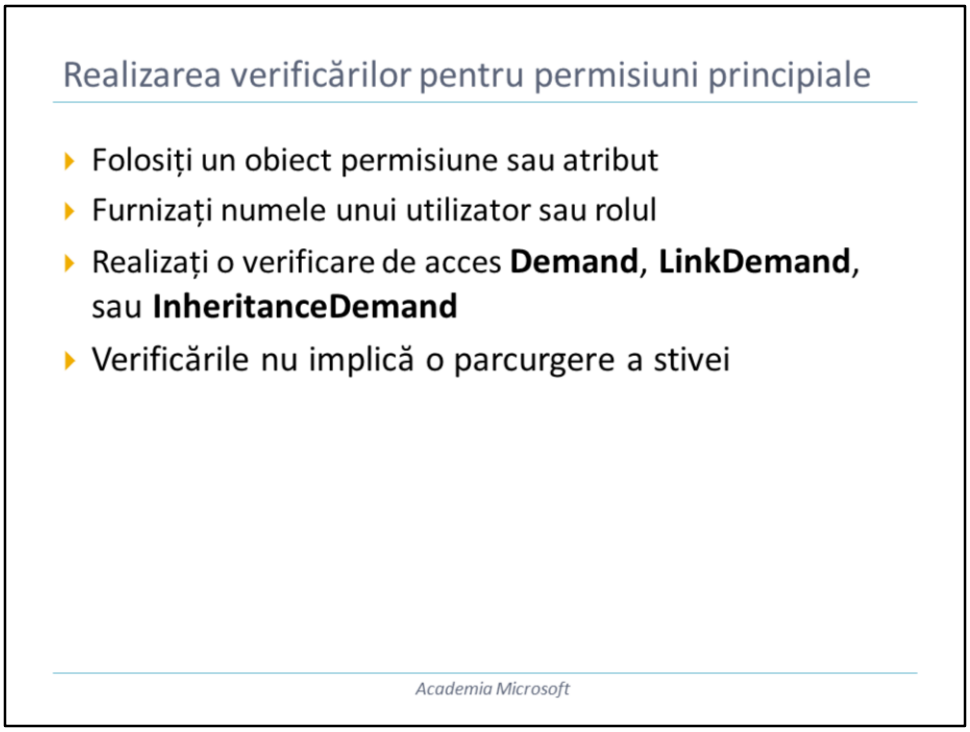
Când creați pentru prima dată un domeniu de aplicație, la runtime va fi atașat principiul Windows curent la firul de execuție prin intermediul metodei **Thread.CurrentPrincipal**. Puteți înlocui principiul de thread prin atașarea unui alt obiect principial. Însă la apelarea acestei proprietăți în cadrul unui proces *.NET Framework*, veți primi implicit un principiu neautentificat și gol. Acest lucru se întâmplă deoarece politica principială pentru un nou domeniu de aplicație este setată implicit la valoarea **PrincipalPolicy.UnauthenticatedPrincipal**.

Dacă doriți ca principiul de thread să fie orice tip de principiu autentificat (spre exemplu **WindowsPrincipal**), trebuie să setați politica principială la valoarea **WindowsPrincipal**, înainte să accesați principiul de thread.

Atenție! Dacă accesați principiul de thread fără a seta politica principială, politica va fi setată la valoarea **UnauthenticatedPrincipal** pentru durata de viață a domeniului de aplicație.

Următorul cod permite principiului de thread să fie folosit de verificări pentru permisiuni principiale, iar principiul inițial va fi un principiu Windows.

```
AppDomain.CurrentDomain.SetPrincipalPolicy(
            PrincipalPolicy.WindowsPrincipal);
Console.WriteLine(Thread.CurrentPrincipal.Identity.Name);
```



Permisiunile principiale verifică principiul curent să fie un utilizator sau un rol specificat. Aceste verificări nu implică o parcurgere a stivei.

Verificări declarative pentru permisiuni principiale

Realizați verificări declarative folosind atributul **PrincipalPermissionAttribute** asupra unei clase sau metode. Când metoda este apelată, la runtime este realizată o verificare de permisiuni asupra principiului pe thread-ul curent, pentru a vedea dacă sunt îndeplinite criteriile specificate de atributul permisiune. Dacă verificarea de permisiune principială eșuează, atunci este ridicat un obiect **SecurityException**, și nu este rulat codul din metodă.

Atenție! Clasa **PrincipalPermissionAttribute** este derivată din clasa **CodeAccessPermissionAttribute**; nu este realizată o parcurgere a stivei.

Acest atribut furnizează trei proprietăți pe care le puteți folosi la verificările de permisiuni. În următorul tabel sunt descrise aceste proprietăți:

Proprietate	Descriere
Authenticated	Acest boolean indică faptul că utilizatorii trebuie să fie autentificați.
Name	Este folosită pentru a furniza numele utilizatorului care va avea permisiune de accesare a metodei.
Role	Este folosită pentru a furniza numele rolului care va avea permisiune de acces a metodei.

În următorul exemplu sunt definite două metode. Prima metodă va oferi acces doar utilizatorilor Windows, care sunt în grupul **Administrator**. Al doilea va oferi acces unui utilizator Windows cu numele **John** pe computerul **MyComputer**.

```
[PrincipalPermission(SecurityAction.Demand,
            Role=@"BUILTIN\Administrators")]
Void OnlyAdministrators()
{
       //code
}

[PrincipalPermission(SecurityAction.Demand,Name=@"MyComputer\John")]
Void OnlyJohn()
{
       //code
}
```

Dacă apelul unei astfel de metode ridică o excepție, puteți obține obiectul **PrincipalPermission** care a eșuat folosind proprietatea **SecurityException.FirstPermissionThatFailed**.

Constructorul **PrincipalPermissionAttribute** are un parametru **SecurityAction**, iar codul precedent folosește o valoare de **Demand**. Alte valori folosite, care sunt permise sunt **LinkDemand** și **InheritanceDemand**, astfel puteți specifica realizarea cererii la compilarea JIT a metodei.

Verificări imperative pentru permisiuni principiale

Realizați comenzi imperative folosind clasa **PrincipalPermission** . Valorile de verificat sunt oferite ca parametrii pentru constructor. Puteți realiza o verificare asupra numelui principial, rolului sau pentru ambele. Următorul cod implementează codul anterior, dar folosește verificări imperative.

```
Void OnlyAdministrators()
{
       PrincipalPermission pp = new PrincipalPermission(null,
            @"BUILTIN\Users");
       pp.Demand();
       //code
}

Void OnlyJohn()
{
       PrincipalPermission pp = new PrincipalPermission
            (@"MyComputer\John",null);
       pp.Demand();
       //code

}
```

Sumar

- Folosirea **Code Access Security**
- Personalizarea modului de autent