OSSIR

« .NET Sécurité »

10 Juin 2002

Eric Mittelette
.NET Developers Evangelist
ericmitt@microsoft.com

Plan de la présentation

- Sécurité .Net = Une Stratégie
  - Managed Code
  - « Evidence based » Security
    - Code Access Security
    - Role Based Security
  - Cryptograhie
- Cas des ASP.NET et WebServices

Introduction

- Le problème :
  - Sécurité de plus en plus complexe
  - Migration d'applications monolithiques vers une approche composants
    - Quid de la sécurité des Transactions, échanges sur Internet…
- Une solution :
  - Une architecture « Managing Software Risk »
    - Pas du tout ou rien, mais un contrôle permanent sur ce que fait le code
  - Basé sur du Code Managé
    - Gestion mémoire sécurisée : prévention par ex des « stack owerflow »

Sécurité sous .NET = combiner un ensemble de mesures ie définir une stratégie.

Exécution Managée

- Commun Language runtime
  - Manager l'exécution des codes .NET
  - Check des droits et permissions
    - Au moment de la JIT Compilation et de l'Exécution
- Class Libraries
  - Jeux de classes « sécurisée »
  - Peut être utilisée sans code spécifique de sécurité
  - Permet de coder « la sécurité » (System.Security, System.Cryptographic)
- Assemblies
  - Dll, exe …(tout code .NET ;-)
    - Metadata et code pouvant contenir des éléments de sécurité

Exécution Managée

- Gestionnaire d' Exceptions unique
  - Au niveau de la CLR, custom
- Buffer « Overrun » controlé par code manager
  - /GS pour l'unmanaged code (prolog et epilog des fonction : canary)
- AppDomains, permet de s'isoler dans un même process
  - Cas de plusieurs roles dans un même process

Exécution Managée

- ON DOIT RESTER VIGILANT:
  - Stratégie de « Storing secrets »
  - Validation des data renforcé (Validators ASP)
  - Désactiver Trace et Debug sur les sites de productions
  - Attention aux Random data
  - Attention a la désérialisation d’objets
  - Gestion de msg d’erreur « brefs » en production

Exécution Managée

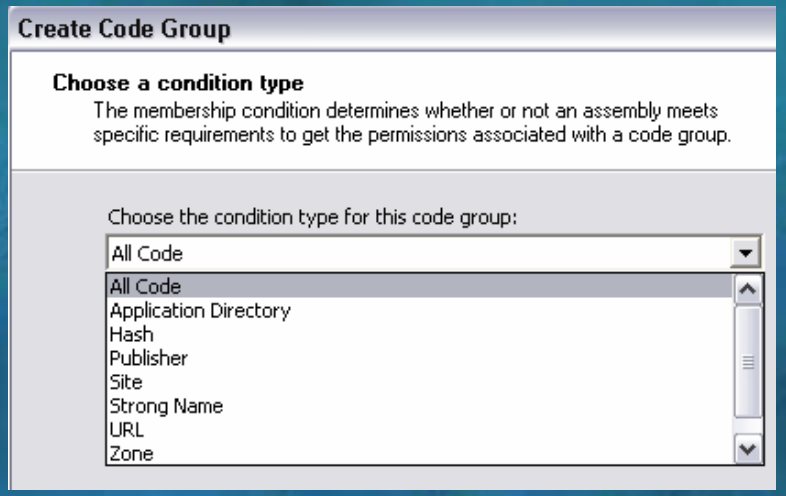
- Démos
  - Buffer overrun and C++
  - Random Number
  - Custom error Handling

« Evidence based » security

- **Stratégie basée sur des « preuves »**
  - **Policy** : Stratégie, .NET donne des permissions qu'au regard de preuves
  - **Permissions** : description de « ressources » et droits associés
    - PermissionRequest : Minimal, Optional et Refuse
    - Toute une liste de permissions « unitaire »
    - **Par exemple Isolated Storage** : droits sur une zone particulière du système de fichier Indépendant des droit IO (sur le système de fichier)
  - **Evidence** : Preuve, Fait..
    - Namespace « scellé » par cryptographie (Strong Name)
    - Identité du « publisher » (Authenticode)
    - Origine du code (URL, Site, IE Zone)

« Evidence based » security

- **Preuves des assembly :**
  - Depuis quel site a été obtenu cette assembly ?
  - Depuis quelle URL ?
  - Depuis quel Zone ?
  - Quel est le « strong name » ?
  - Qui l'a signé ?



« Evidence based» security

- **Démos**
  - **SN et assembly**
  - **Certificat digital**
  - **Zone et droits par défaut**

Code Access Security

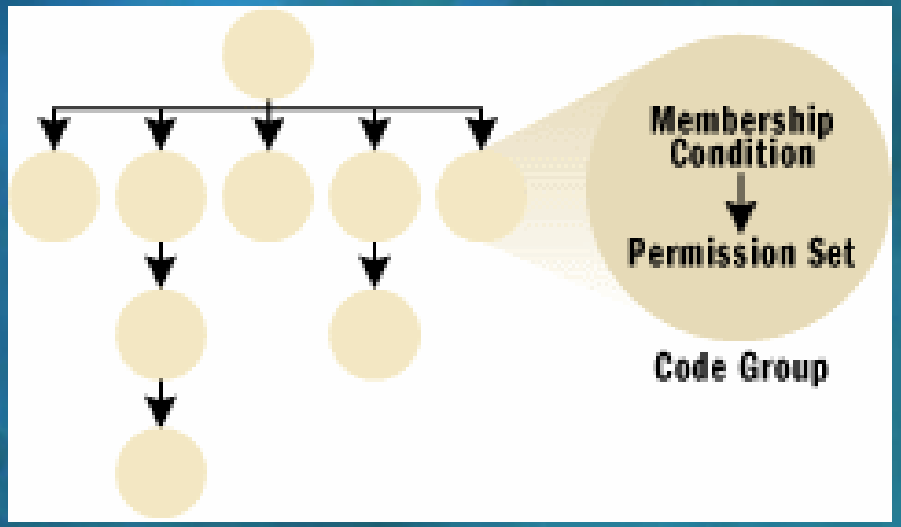
- Mécanisme permettant de garantir que l'application n'a pas plus de droits que ceux attribués

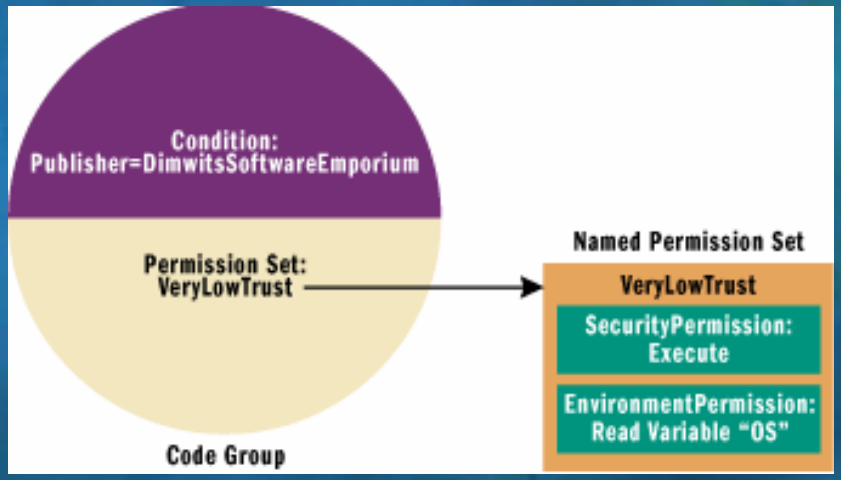
- Permet aux code .NET de faire une demande sur une ressource donnée

- **Imperative** : contrôles au runtime
- **Declarative** : sous forme de metadata, certains contrôle sont effectués à la compilation, chargement…

Code Access Security

- .NET runtime gère une arborescence de code group
- Un chemin dans cet arbre = une stratégie

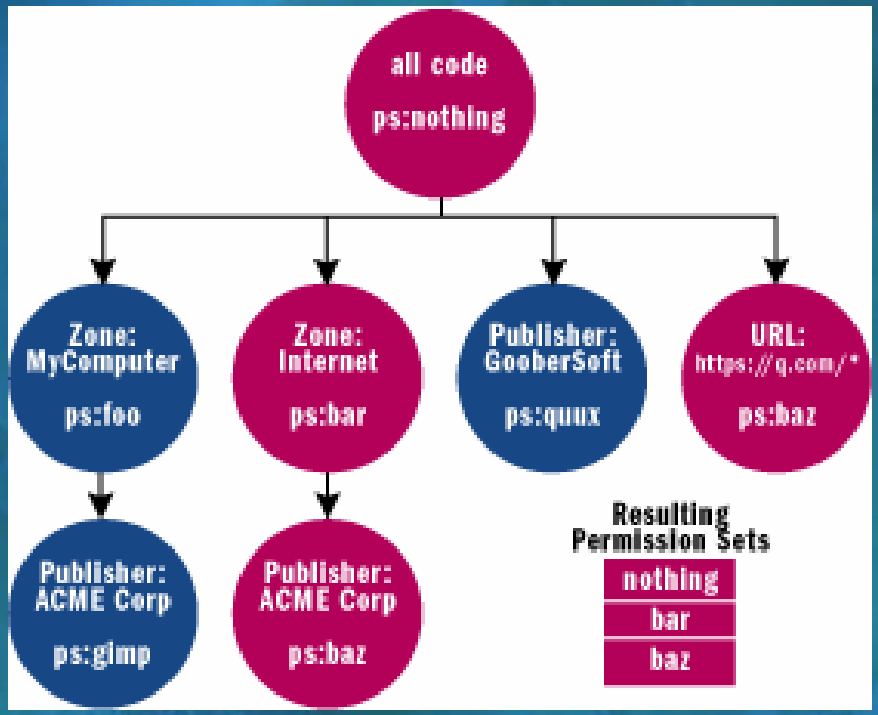


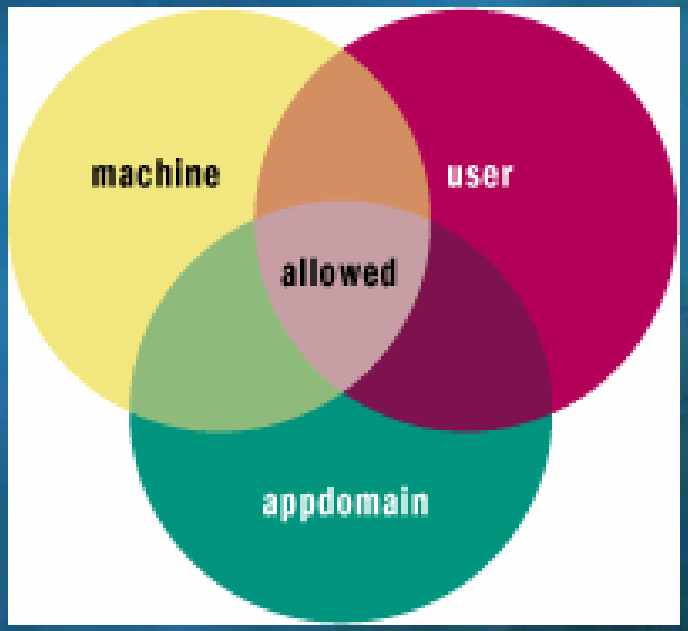


- **Code Group**
  - Possède :
    - Un « Membership Condition »
    - Un « PermSet »

Code Access Security

- Parcours de l'arbre
- « Stack Walk »
  - Par exemple on accède à:
  - https://q.com/downloads/foobar.dll
  - signed by ACME Corporation
  - Quels seront les droits ?



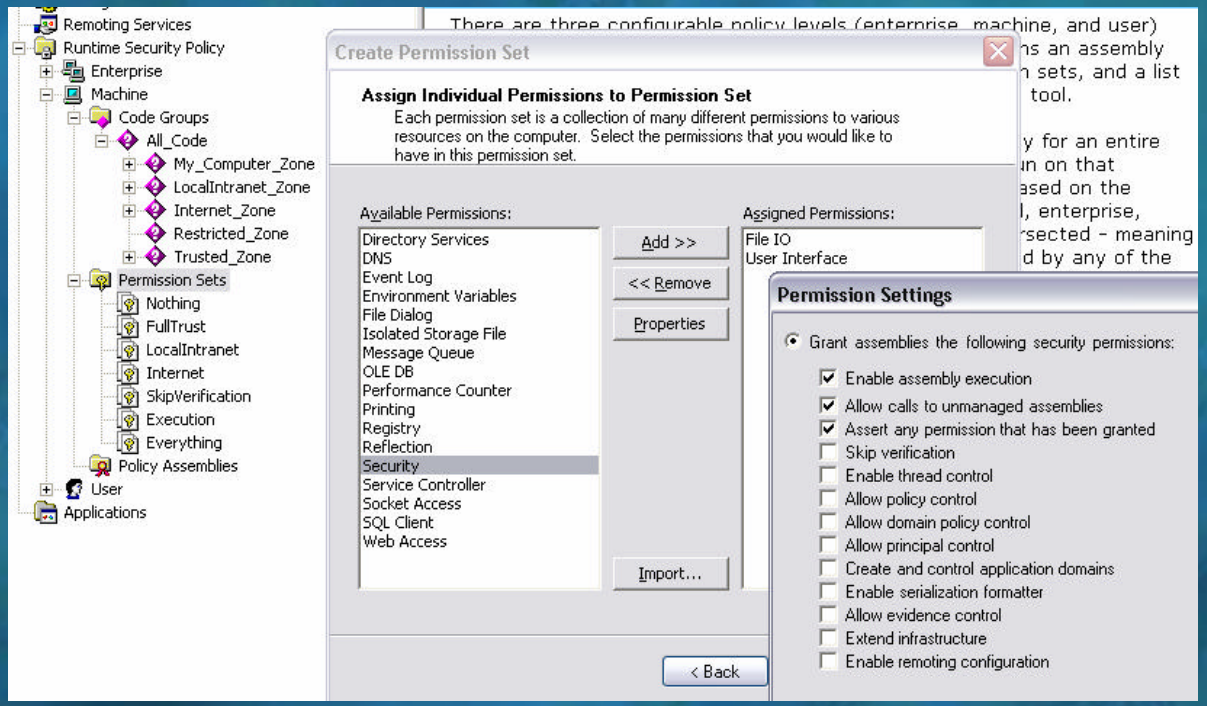


- Permset résultant = Intersection des 3 niveaux

Code Access Security

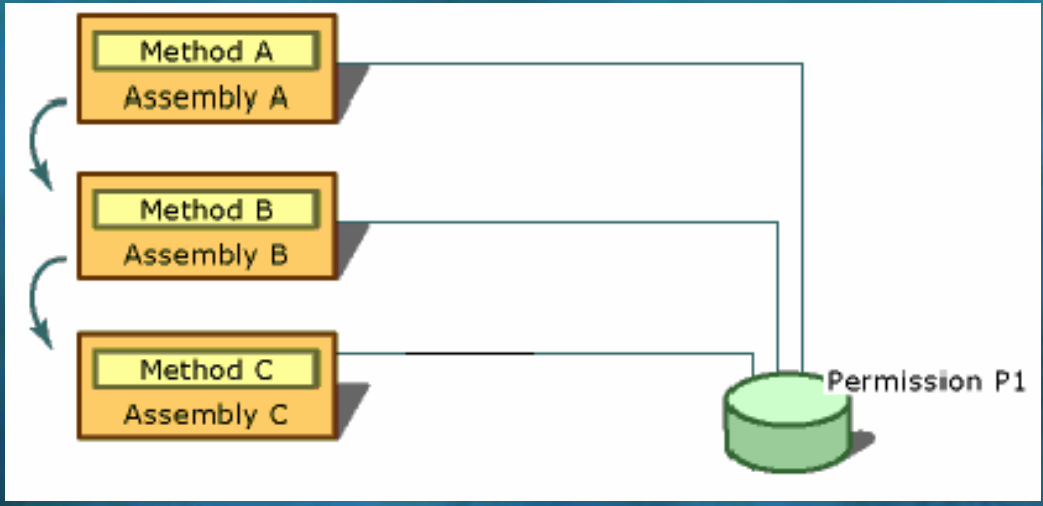
- Permissions « Unitaires »
  - .NET définit un ensemble fini de permissions

Directory Services
DNS
Event Log
Environment Variables
File IO
File Dialog
Isolated Storage File
Message Queue
OLE DB
Performance Counter
Printing
Registry
Reflection
Security
Service Controller
Socket Access
SQL Client
Web Access
User Interface



Code Access Security

- Notion d'héritage de droits entre appelant
  - A appel B, B accède a une ressource
  - B a les droits, A ne les a pas
  - A peut il accèder aux ressources ?



Code Access Security

- Prenons un exemple : Un composant de trace

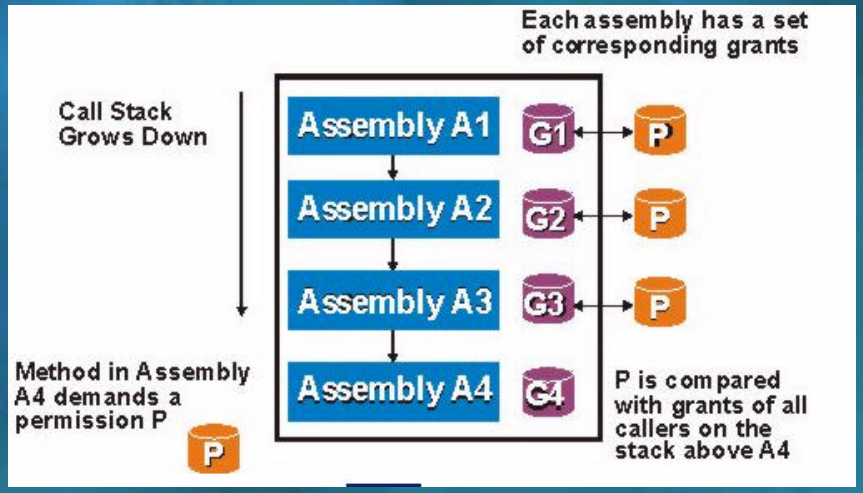
```
public class ErrorLogger
    {
        public static void Log(String s)
        {
            const String fname = "c:\\temp\\errlog.txt";
            FileStream logStream = new FileStream(fname,FileMode.Append);
            StreamWriter logWriter = new StreamWriter(logStream);
            logWriter.Write(s);
            logWriter.Close();
            logStream.Close();
        }
    }
```

Demande d'accès explicit a une ressource fichier

```
public class ErrorLogger1Bis
{
    public static void Log(String s)
    {
        const String fname = "c:\\temp\\errlog.txt";
        FileIOPermissionAccess desiredAccess = FileIOPermissionAccess.Append;
        FileIOPermission p = new FileIOPermission(desiredAccess, fname);
        p.Demand();
        FileStream logStream = new FileStream(fname,FileMode.Append);
        StreamWriter logWriter = new StreamWriter(logStream);
        logWriter.Write(s);
        logWriter.Close();
        logStream.Close();
    }
}
```

Code Access Security

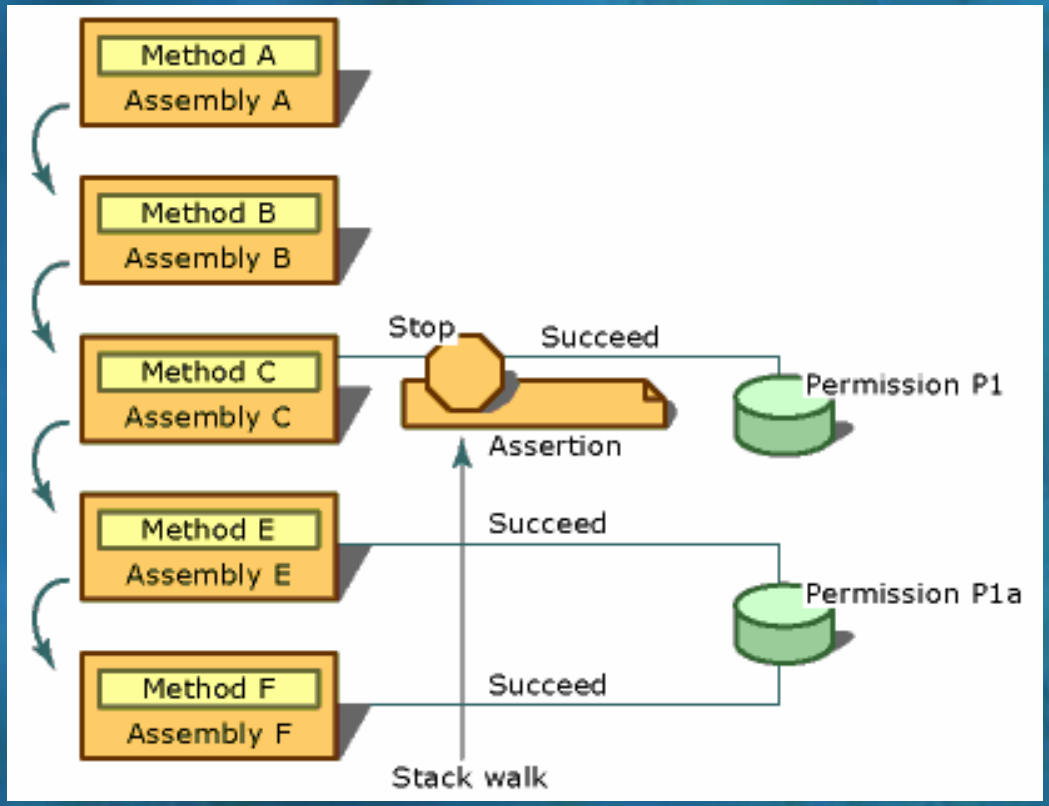
- Mécanisme de Stack Walk déclenché par :
  - Demand()
    - (Declarative au JIT)
  - Appel « intercepté»
    - (Imperative, au runtime)



- Modification du parcours de pile
  - On a les droits mais on interdit tout appel (prévention)
    - Deny()
  - Je me porte garant pour tout appel vers une ressource (il faut que l'assembly est elle-même les droits)
    - Assert()

Code Access Security

- Assert et stack call



Code Access Security

- Assertion : Engager sa responsabilité
  - Impérative

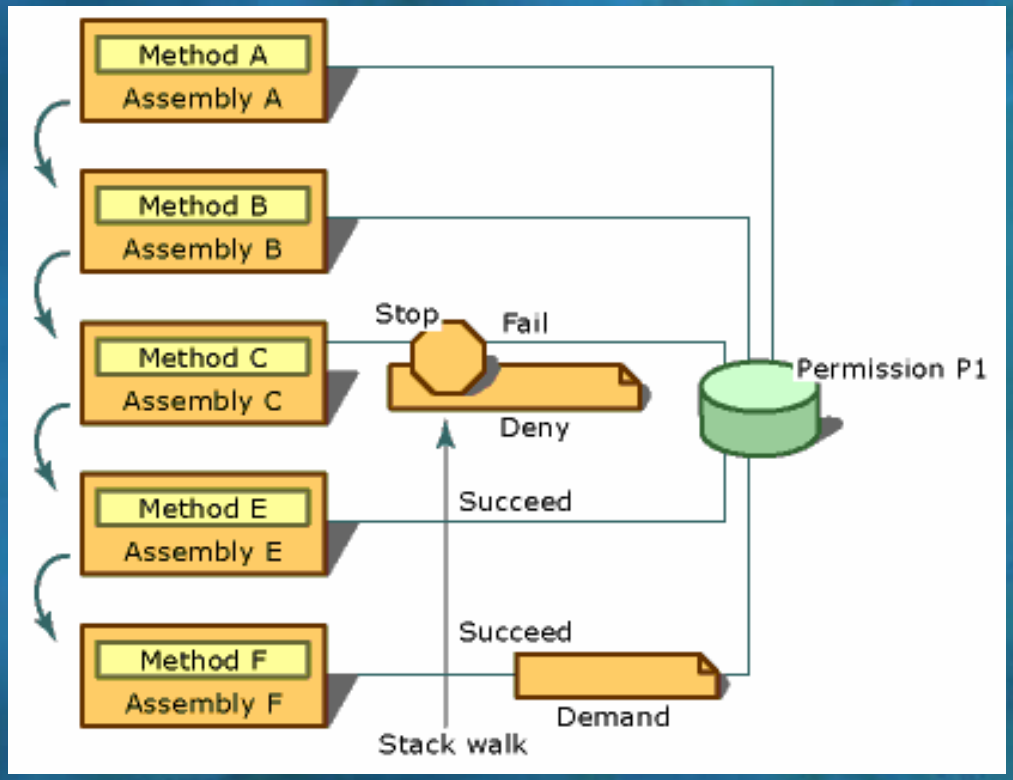
```
public class ErrorLogger2
{
    public static void Log(String s)
    {
        const String fname = "c:\\temp\\errlog.txt";
        FileIOPermission p = new FileIOPermission(FileIOPermissionAccess.Append, fname);
        p.Assert();
        FileStream logStream = new FileStream(fname,FileMode.Append);
        StreamWriter logWriter = new StreamWriter(logStream);
        logWriter.Write(s);
        logWriter.Close();
        logStream.Close();
    }
}
```

  - Déclarative

```
public class ErrorLogger3
{
    const String fname = "c:\\temp\\errlog.txt";
    [FileIOPermission(SecurityAction.Assert, Append=fname)]
    public static void Log(String s)
    {
        FileStream logStream = new FileStream(fname,FileMode.Append);
        StreamWriter logWriter = new StreamWriter(logStream);
        logWriter.Write(s);
        logWriter.Close();
        logStream.Close();
    }
}
```

Code Access Security

- Deny et stack call



Code Access Security

- Deny : les futurs appels sont interdits
  - Imperative

```
String fname = @"c:\temp\errlog.txt";
FileIOPermission p = new FileIOPermission(FileIOPermissionAccess.Write,fname);
p.Deny();
```

  - Declarative

```
private const String path = @"c:\temp\errlog.txt";
[FileIOPermission(SecurityAction.Deny,Write=path)]
static void Test()
{
    try
    {
        FileIOPermission p = new FileIOPermission(FileIOPermissionAccess.Write,path);
        p.Demand();
    }
    catch(SecurityException)
    {
        Console.WriteLine("Demand Failed");
    }
}
```

Assembly Permissions Requests

- Les assemblies peuvent faire 3 types de requêtes a la CLR (au chargement)
  - RequestMinimun
    - L'Assembly ne sera pas chargée si elle ne dispose pas de ces permissions
  - RequestRefuse
    - L'Assembly ne recevra jamais ces droits
  - RequestOptional
    - L'Assembly sera chargée, mais pb possible
- Les requêtes ne permettent pas d'obtenir plus de droits qu'autorisés…

Code Access Security

- Démos
  - Imperative
  - Declarative
  - Assert
  - Unmanaged
  - Demand, deny…

Role Based Security

- Notion d’ Identificateur et de Rôles
  - Comparable au mécanisme COM+
- Role et users peuvent être des comptes/groupes NT
- Role Based Security est traité au même niveau que Code Access Security
- Utilise l’Objet : PrincipalPermission

Role Based Security

- L'objet Principal decrit un user et un rôle
  - WindowsPrincipal
  - GenericPrincipal
- CurrentPrincipal : propriété du thread courant (Get/Set)

```
String id1 = "bob";
String role1 = "Manager";
PrincipalPermission perm1 = new PrincipalPermission(id1,role1);

String id2 = "luc";
String role2 = "Developer";
PrincipalPermission perm2 = new PrincipalPermission(id2,role2);

(perm1.Union(perm2)).Demand();
```

Role Based Security

- **Generic principal**
  - Cas des authentifications hors base de compte NT
  - Création, puis attache sur le thread courant

```
//Create generic identity.
GenericIdentity MyIdentity = new GenericIdentity("MyIdentity");

//Create generic principal.
String[] MyStringArray = {"Manager", "Teller"};
GenericPrincipal MyPrincipal = new GenericPrincipal(MyIdentity, MyStringArray);

//Attach the principal to the current thread.
Thread.CurrentPrincipal = MyPrincipal;
```

Role Based Security

- **WindowsPrincipal**
  - Information sur l'utilisateur connecté. (au sens Windows)

```
//Get the current identity and put it into an identity object.
WindowsIdentity MyIdentity = WindowsIdentity.GetCurrent();

//Put the previous identity into a principal object.
WindowsPrincipal MyPrincipal = new WindowsPrincipal(MyIdentity);
```

Récupérer le windowsprincipal

Accès aux propriétés du WindowsPrincipal

```
//Principal values.
string Name = MyPrincipal.Identity.Name;
string Type = MyPrincipal.Identity.AuthenticationType;
string Auth = MyPrincipal.Identity.IsAuthenticated.ToString();

//Identity values.
string IdentName = MyIdentity.Name;
string IdentType = MyIdentity.AuthenticationType;
string IdentIsAuth = MyIdentity.IsAuthenticated.ToString();
string ISAnon = MyIdentity.IsAnonymous.ToString();
string IsG = MyIdentity.IsGuest.ToString();
string IsSys = MyIdentity.IsSystem.ToString();
string Token = MyIdentity.Token.ToString();
```

Role Based Security

- Le Principal contient la méthode « IsInRole »
  - Permet de tester l'appartenance a un groupe ou un rôle

```
WindowsIdentity wi = WindowsIdentity.GetCurrent();

WindowsPrincipal wp = new WindowsPrincipal(wi);

if (wp.IsInRole(WindowsBuiltInRole.Administrator))
    Console.WriteLine(WindowsBuiltInRole.Administrator.ToString() + " Admin");

if (wp.IsInRole("BUILTIN\\Administrators"))
    Console.WriteLine("BUILTIN\\Administrators : " + wp.ToString());
```

Role Based Security

- Démos:
  - Role Based Security
    - Generic, permission et Principal

Cryptographie et .NET

- **La Cryptographie a plusieurs buts:**
  - **Confidentialité**: Masquer l'identité utilisateur ou des données.
  - **Intégrité des données**: Protéger les données de toute altérations.
  - **Authentification**: Assurer que des données sont bien issues d'une source donnée.

- **Les Services Cryptographiques sous .NET**
  - Implémentation Managée
  - et/ou wrappée sur CAPI (Crypto API)
  - Un namespace principal :
    - System.Security.Cryptography

Cryptographie et .NET

- Plusieurs namespaces consacrés à la cryptographie dans le Framework .NET
  - **System.Security.Cryptography**
    - Fournis les services cryptographiques :
      - Encodage/Décodage sécurisé des données
      - hashing, random number, message authentication. …
  - **System.Security.Cryptography.X509Certificates**
    - Authenticode X.509 v.3 certificate.
    - Le certificat est signé avec une « private key » qui identifie explicitement et de manière unique le propriétaire du certificat.
  - **System.Security.Cryptography.Xml**
    - Réservé a données XML
    - Permet de signer les « objets » XML avec une signature digitale

Cryptographie et .NET

- Model objet du namespace **Cryptography**
  - **Algorithme Type** : (abstract class)
    - Algorithmes Symétrique et Hash
  - **Algorithme Class** : (abstract class)
    - RC2, SHA1…
  - **Implémentation des « Algorithmes Class »**
    - RC2CryptoServiceProvider, SHA1Managed…
  - Grâce a ce « design pattern » il est possible de faire ses propres ajouts/implémentations aux algorithmes aux différents niveaux.
  - Les algorithmes symétriques et de hash sont implémentés « stream-oriented »
  - La Configuration Cryptographique
    - Permet de résoudre les implémentations spécifiques de certains algorithmes

Cryptographie et .NET

- Primitives standards disponibles dans le Framework .NET
  - RSA et DAS (public key encryption, asymetric)
  - DES TripleDES RC2 (private key encryption, symmetric)
  - MD5 et SHA1 (hashing)

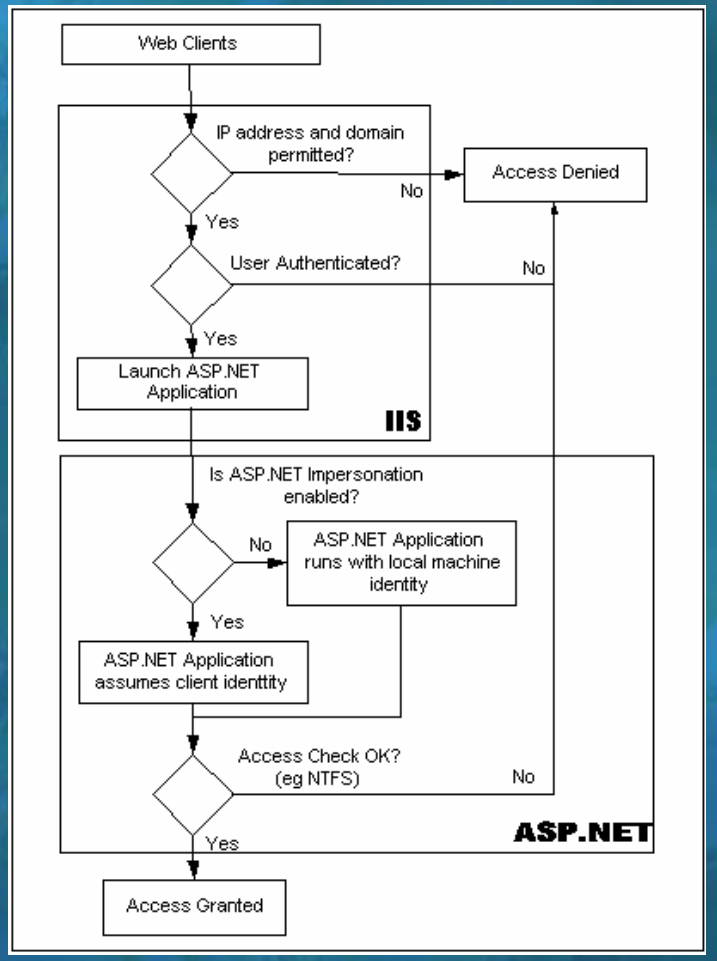
  - Digital signature
  - MACs

Cryptographie

- Démos
  - Cryptage symétrique
  - Base Class

ASP Security

- Relation entre IIS et ASP.NET
  - ASP.NET s'appuie sur IIS
  - Spécialise la configuration



Authentification et Autorisation

- Authentification IIS
  - Anonyme
  - Basic/Digest
  - Integrated
- Authentification ASP.NET
  - Config.web
  - Windows, Passport, Forms, None

```
<authentication mode="Windows"/>
<identity impersonate="true"/>
```

```
<authentication mode="Forms">
<forms name="TestAuthen"
       loginUrl="login.aspx"
       protection="All" >
</forms>
</authentication>
```

```
<authentication mode="Passport">
    <passport redirectUrl="PassportURL"/>
</authentication>
```

- Autorisation (web.config)
  - allow
  - deny

```
<authorization>
      <allow users="Kim"/>
      <deny users="John" />
      <deny users="?" />
</authorization>
```

```
<authorization>
      <allow verb="GET" users="*" />
      <allow verb="POST" users="Kim" />
      <deny verb="POST" users="*" />
</authorization>
```

```
<configuration>
   <system.web>
      <authorization>
         <allow roles="Admins" />
         <deny users="*" />
      </authorization>
   </system.web>
</configuration>
```

Web Services

- Modes d’ Authentification
  - Windows - Basic
  - Windows - Basic over SSL
  - Windows – Digest
  - Windows - Integrated Windows
  - Windows – Client Certificates
  - SOAP headers – Custom

– Pas de forms…

Web Services

- Passer les « credentials » a l'appel

```
// Create a new instance of CredentialCache.
CredentialCache credentialCache = new CredentialCache();

// Create a new instance of NetworkCredential using the client
// credentials.
NetworkCredential credentials = new
    NetworkCredential("username","password","domain");

// Add the NetworkCredential to the CredentialCache.
credentialCache.Add(new Uri(math.Url),
                    "Basic", credentials);

// Add the CredentialCache to the proxy class credentials.
math.Credentials = credentialCache;
```

```
// Load the client certificate from a file.
X509Certificate x509 = X509Certificate.CreateFromCertFile(@"c:\user.cer");

// Add the client certificate to the ClientCertificates property
// of the proxy class.
bank.ClientCertificates.Add(x509);
```

Utilisation d'un certificat x509

ASP.NET

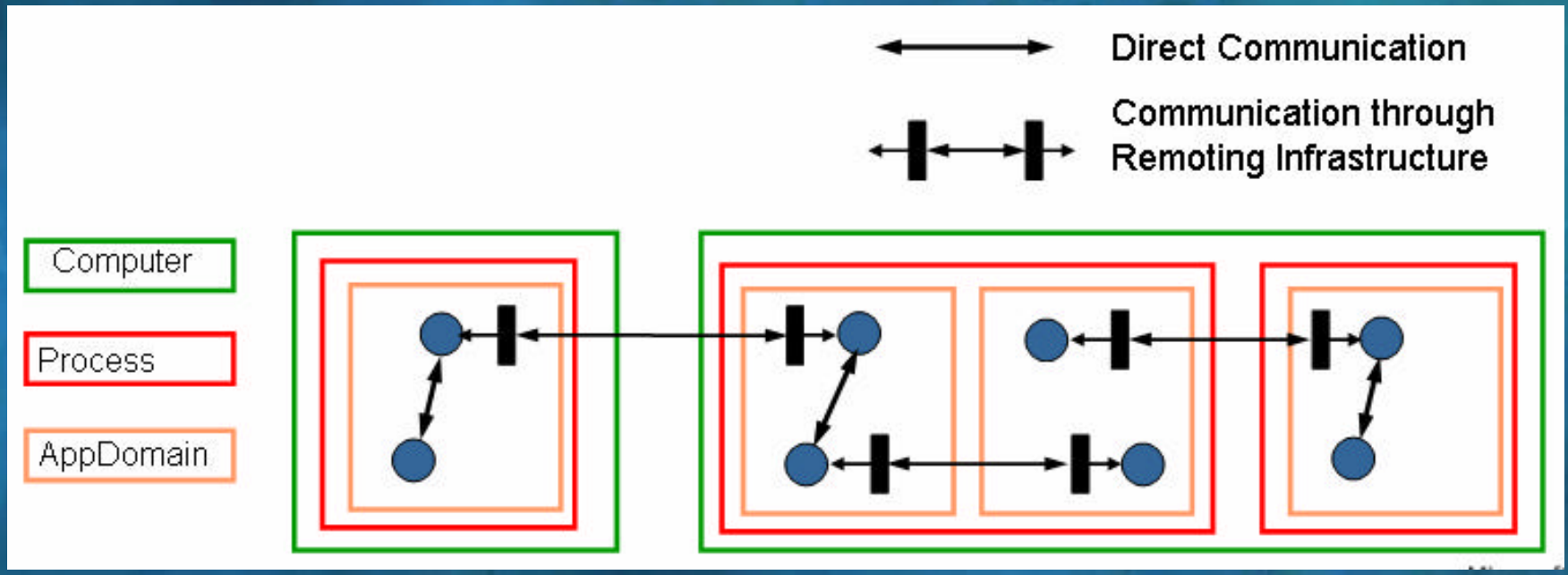
- Démos
  - Authentification
    - Windows, Form
  - WebServices

AppDomains

- Notion comparable a un process
  - Il peut y a avoir plusieurs « appDomain » dans un process Win32
  - Isolation des « principals » entre appDomain d'un même process
  - C'est sur appDomain que s'applique:
    - SetAppDomainPolicy
    - SetPrincipalPolicy
    - SetThreadPrincipal

AppDomains

- Seuls les objets d'un même « AppDomain » peuvent communiquer directement.
- Toute autre communication .NET utilise le Remoting.
- La securité est appliquée par appDomain



AppDomains

- Démos
  - Création et gestion d’un appDomains