

# Building Secure Distributed Applications with .NET

**Dominick Baier (dbaier@ernw.de)**  
**.NET Security Consultant / BS 7799 Lead Auditor**

**[www.leastprivilege.com](http://www.leastprivilege.com)**



# Outline

- **Threats and Mitigation & General Principles**
- **Architecture and Mechanics**
  - Authentication
  - Authorization
  - Input Validation
  - Windows Security
  - Code Access Security
  - Instrumentation
- **Security in the**
  - Presentation Layer
  - Business Layer
  - Data Layer
- **Communication between Layers**



# Threats and Mitigation

- **Objectives**

- What types of threats are out there?
- Ways of mitigating those threats
- A process for designing secure code
- Some guiding principals for writing secure code
- Recommended reading



# The STRIDE Threat Model

- **STRIDE**
  - **S**poofing Identity
  - **T**ampering with data
  - **R**epudiation
  - **I**nformation Disclosure
  - **D**enial of Service
  - **E**levation of Privilege



# Spoofing identity

- **Attacker pretends to be someone he is not**
  - there are two flavors of this attack
- **Spoofing client identity**
  - access a server and pretend to be a legitimate user
  - gain access to sensitive data
  - run potentially dangerous queries/processes on the server
  - gain administrative access to the server
- **Spoofing server identity**
  - pretend to be a legitimate server to unsuspecting clients
  - collect sensitive data from clients
  - provide false data to clients
  - often opens the door for other attacks



# Mitigating the spoofing threat

- **Strong authentication is the best defense**
  - authentication is a secure process for validating identity
  - clients can prove their identities to servers
  - servers can prove their identities to clients
- **Identity can be proved in several ways**
  - something you have
  - something you know
  - something you are
- **Authenticating over a network requires cryptography**
  - more on this later...



# Tampering with data

- **Attackers often gain advantage by tampering with data**
- **Tampering with persistent data**
  - change prices for products they want to buy online
  - modify audit logs to cover their tracks
  - modify password data to gain access to other user accounts
  - corrupt data files to crash, or even take over a server
  - deface web pages
  - add viruses or Trojan horses to files
  - tamper with network topology (routing tables, DNS, etc.)
- **Tampering with network packets**
  - tampering with packets on the wire



# Mitigating the tampering threat

- **Protect persistent data**
  - hash codes
  - digital signatures
  - encryption
  - example: NTFS Encrypting File System (EFS)
- **Protect network packets**
  - network authentication protocols usually offer integrity and confidentiality protections via cryptography
  - Kerberos
  - SSL/TLS
  - IPSec



# Repudiation

- **Attacker denies an action, and victim cannot prove otherwise**
  - an attacker might:
    - claim he didn't delete a file
    - claim he didn't make a purchase or return
    - claim he didn't receive goods/services



# Mitigating the repudiation threat

- **Mitigation techniques are called non-repudiation**
  - audit actions in the OS and protect the audit logs
  - require receipts as acknowledgement
  - use timestamps
  - digital signatures can help with electronic transactions



# Information disclosure

- **Attacker sees data he shouldn't be seeing**
  - local files
  - data traveling between computers
- **Attacker sees information about the system architecture**
  - banners that display software type and version
  - helps the enemy narrow down potential attacks



# Mitigating the information disclosure threat

- **Use strong authentication and consistent access control**
- **Encryption might help**
  - NTFS EFS, for example
- **Turn off banners on publicly exposed services**
  - or expose purposely misleading banners
  - obscurity is not security but sometimes it helps
- **Disable tracing and debugging features in production apps**
- **Avoid sending verbose error information to clients**
  - pipe this information to internal logs instead



# Denial of service (DoS)

- **Attacker causes your service to become unavailable**
  - usually associated with services provided over a network
    - syn flood attacks
    - distributed denial of service attacks (DDoS)
    - amplification attacks such as smurf
    - all designed to consume precious bandwidth!
  - the anonymity of TCP/IP doesn't help
    - DoS attacks are quite troublesome because the attacker can spoof his source ip address randomly



# Mitigating the denial of service attack

- **Increase availability and reliability**
  - make sure your system doesn't melt under high loads
  - have a strategy for throttling requests
  - consider clustering
  - buy more bandwidth
- **Filtering and throttling**
  - block incoming ICMP broadcasts (for example)
  - throttle anonymous requests
- **Be a good neighbor**
  - egress filtering (verify source IP addr on outgoing packets)
  - automate virus checking to avoid DDoS zombies



# Elevation of privilege

- **Attacker finds a way to gain more privileges on the system**
  - the ultimate goal is to gain administrative privileges
  - most common exploit is the buffer overflow (more on this later)
  - bugs in the operating system itself can allow this



# Mitigating the elevation of privilege threat

- **Produce and consume only quality, robust code**
  - avoid common security errors like buffer overflows
  - fear user input
    - more on this later
  - run code with only the privileges it really needs
    - known as *Principal of Least Privilege*
  - eliminate dead code
    - code paths that are never used
    - features of third party software that you don't need
  - keep up to date with the latest operating system patches
    - HFNETCHK.EXE + Baseline Security Analyzer
    - <http://www.microsoft.com/security>

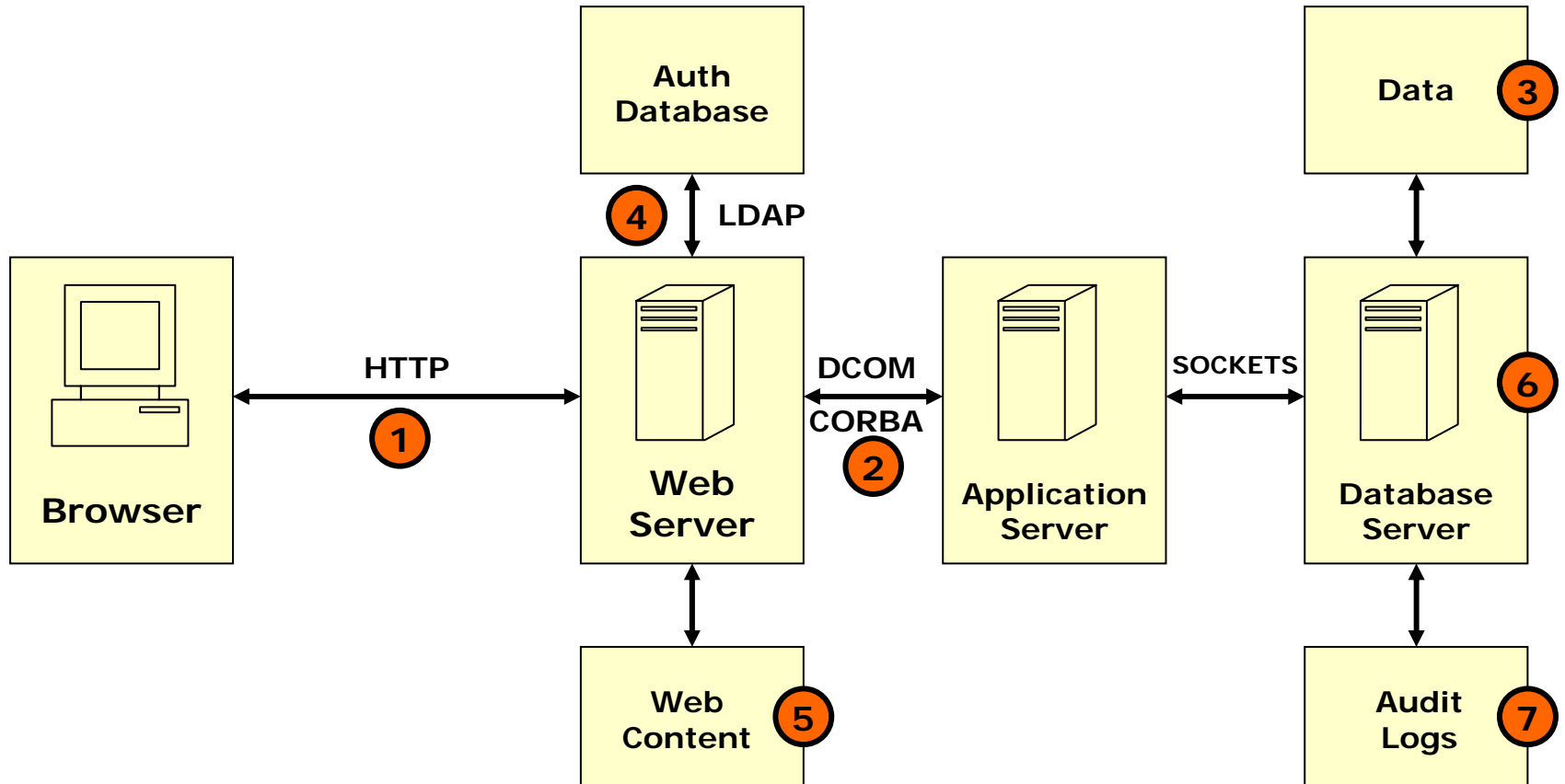


# Summary of STRIDE threats and mitigation

- **STRIDE**
  - **Spoofing Identity**
    - strong authentication
  - **Tampering with data**
    - hash codes, digital signatures, encryption
  - **Repudiation**
    - audit logs, receipts, digital signatures, timestamps
  - **Information Disclosure**
    - strong authentication, access control, encryption, obscurity
  - **Denial of Service**
    - increase availability, reliability, and be a good neighbor
  - **Elevation of Privilege**
    - robust code, least privilege, OS patches



# STRIDE in the real World



1-3: T/I --- 4: S/I/E --- 5-6: T --- 7: T/R --- 8: D/S



# The three components of a secure system

- **Just as with physical security, we need all three**
  - protection
  - detection
  - reaction
- **You don't need unbreakable protection**
  - you really can't achieve this anyway
  - many developers throw up their hands if they can't design a perfect solution (it feels frustrating)
- **Design detection and reaction into your systems**
  - protection then becomes a way to slow down the attacker
  - once detected, an attack can be halted by a sysadmin
  - diagnose and patch the problem quickly



# A process for developing secure apps

- **Security should be an integral part of the design process**
  - write down your security goals
  - examine the system architecture
  - determine the threats using STRIDE
  - prioritize threats
    - $\text{risk} = (\text{potential damage}) \times (\text{likelihood of success})$
  - choose a response
    - accept the risk as is
    - warn the user (transfer the risk)
    - remove the feature (remove the risk)
    - fix the problem (mitigate the risk)
  - revisit your security strategy with each iteration!



# General principals to live by

- **Security is a feature**
- **Use least privilege**
- **Layer your defenses**
- **Pay attention to failure modes**
- **Prefer secure defaults**
- **Cryptography doesn't ensure security**
- **Firewalls don't ensure security**



# Security is a feature

- **Security is a crosscutting feature**
  - Similar to performance
- **Impossible to bolt on security at the end of a project**
  - Requires constant attention and iteration
- **Be sure you have a security feature team**
- **Need to convince management you need security?**
  - It's amazing what a demonstration can do



# Use least privilege

- **Run your code with only the privileges it requires**
  - most services don't need to run as SYSTEM
  - most desktop apps don't need admin privileges
- **Use WinXP and .NET Server's built in low-privilege accounts**
  - NT Authority \ LocalService
  - NT Authority \ NetworkService
- **Don't be lazy**
  - open kernel objects for only the permissions you really need
  - test your code in a non-administrative environment
  - or go one step further and WRITE your code in a non-administrative environment!



# Layer your defenses

- **Don't assume someone else will save you**
  - Consider your code the last bastion of defense
  - Validate input data



# Pay attention to failure modes

- **Developers focus on normal paths of execution**
- **Attackers focus on failure modes**

*devote at least as much time to design, code,  
and test error handling paths as you do for  
normal paths of execution*



# Prefer secure defaults

This is real bad!!

The installation is complete. For your convenience, all ports have been enabled and an administrator account has been created. The password is "password". If you want to secure your application, follow these 15 easy steps



# Cryptography doesn't ensure security

- **So what if you're using 128-bit encryption?**
  - Are the keys generated from low entropy passwords?
  - Where are the keys stored?
  - How are the keys exchanged?
  - How much data is encrypted with a single key?
  - Did you realize that encryption does not ensure integrity?



# Firewalls don't ensure security

- **Any application exposed through the firewall must be robust**
  - Dumb code in exposed applications leads to compromise
- **How do you manage trust across a firewall?**
  - Do you trust the authority outside the firewall to authenticate external clients?



# Books every Windows programmer should own

- **Secrets and Lies**
  - Schneier
- **Hacking Exposed (latest edition)**
  - McClure, et al.
- **Writing Secure Code**
  - Howard & LeBlanc
- **.NET Framework Security**
  - LaMacchia, et al.
- **The .NET Developers Guide to Windows Security**
  - Brown



# Architecture and Mechanics

- **Objectives**
  - Application Architecture and Layers
  - Authentication
  - Authorization
  - Input Validation
  - Windows Security
  - Code Access Security
  - Instrumentation



# Security on each Layer

Secure Communication

## User Interface

How do I customize the user interface based on a user's role?  
How do I validate user input?  
Does the Front-End run with Least Privilege / Partial Trust?  
How do I store secrets?

## Business Logic

How can I write code to manage authorization in business processes?  
How do I make sure I know what identity to authorize on in server environments?  
How should I build reaction and detection in the Business Tier  
Does the Business Tier run with Least Privilege / Partial Trust?

## Data Access

How do I prevent sensitive data from reaching unauthorized users?  
How do I ensure anonymous users don't modify data?  
How can I store sensitive data?



# What is?

- **Authentication**

- The process of identifying a principal
- A principal can be a user, a computer, a network device or an assembly
- The principal is identified through credentials

- **Authorization**

- Is the confirmation that an authenticated principal has permissions to perform a specific operation
- Protection allows only appointed users to perform certain actions, and it prevents malicious acts



# How to Authenticate?

- **Windows Built-In Authentication**
  - NTLM / Kerberos
  - ASP.NET Middle Tier
  - COM+ Middle Tier
- **Custom Authentication**
  - ASP.NET Forms Authentication
  - Custom Application Logins
- **Translate between the two**
  - Impersonation
  - Kerberos Protocol Transition

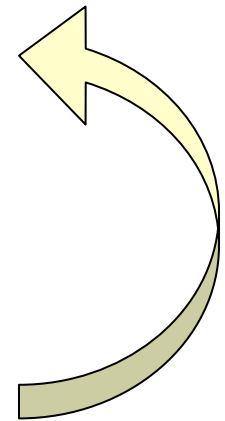


# Using Role-Based Security in .NET Applications

- .NET provides two interfaces for Role-Based Security in the System.Security.Principal Namespace
- Decouples Authentication and Authorization

```
interface IIdentity {  
    bool    IsAuthenticated    { get; }  
    string  AuthenticationType { get; }  
    string  Name                { get; }  
}
```

```
interface IPrincipal {  
    IIdentity Identity { get; }  
    bool      IsInRole(string roleName);  
}
```



# Use Roles for Authorization

- **A Role is a category or a set of users who share the same security privileges**
  - Easier to use than authorizing individual users
  - Memberships can change without having to adjust the logic
- **Roles based on the business organization**
  - Manager
  - Employee
  - Customer
- **Roles based on the types of operations**
  - CanDeleteCustomer
  - CanApproveClaim
  - CanShutDownNuclearReactor



# Identities

- The .NET Framework provides four classes that implement the **Identity** interface
  - WindowsIdentity
  - GenericIdentity
  - PassportIdentity
  - FormsIdentity

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



# Principals

- The **IPrincipal** interface links user roles and identities
- Linking the **IPrincipal** object to the thread provides easy access, and you don't have to carry it around
  - Plumbing does this (e.g. ASP.NET)

```
class Plumbing {  
    public static void DoHeavyLifting() {  
        IPrincipal client = AuthenticateUserSomehow();  
        Thread.CurrentPrincipal = client;  
        Application.ImplementBusinessLogic();  
    }  
}
```



# Performing Role Check

- The `IPrincipal.IsInRole()` Method

```
class Application {  
    public static void ImplementBusinessLogic() {  
        if (Thread.CurrentPrincipal.IsInRole("Managers")) {  
            // do something privileged  
        }  
    }  
}
```



# Performing Role Checks - Declaratively

- **PrincipalPermissionAttribute uses Thread.CurrentPrincipal**
  - allows you to add declarative checks against client identity
  - JIT compiler inserts the checks at the start of the method

```
using System.Security.Permissions;

[PrincipalPermission(SecurityAction.Demand, Authenticated=true)]
class PetStore {
    public void PetAnimals() { ... }
    public void BuyAnimals() { ... }

    [PrincipalPermission(SecurityAction.Demand, Role="Staff")]
    public void FeedAnimals() { ... }

    [PrincipalPermission(SecurityAction.Demand, Role="Managers")]
    public void GiveRaise() { ... }
}
```



# Performing Role Checks - Imperatively

- The same as calling `IsInRole()`, but will generate a `SecurityException` if demand fails

```
try {  
    PrincipalPermission OperatorPermission =  
        new PrincipalPermission(null, "Operator");  
  
    OperatorPermission.Demand();  
}  
catch (SecurityException se) {}
```



# Performing Role Checks - Imperatively

- Implementing an OR scenario

```
PrincipalPermission OperatorPermission =  
    new PrincipalPermission(null, "Operator");  
PrincipalPermission TechnicalStaffPermission =  
    new PrincipalPermission(null, "TechnicalStaff");  
  
(OperatorPermission.Union(TechnicalStaffPermission)).Demand();
```

- Implementing an AND scenario

```
OperatorPermission.Demand();  
TechnicalStaffPermission.Demand();
```



# Performing Authorization based on Windows Authentication

- **Windows uses NTLM or Kerberos to validate credentials**
- **A successful authentication produces a Windows token**
- **Link this Token to Thread.CurrentPrincipal**
  - Manual or with `AppDomain.PrincipalPolicy`
- **Windows Roles have the Format : AuthorityName\GroupName**
  - Watch out for localized names
  - Use the `WindowsBuiltInRole` enumeration
    - `NT Authority\Network Service == NT Autorität\Netzwerk Dienst`
  - Use `Environment.UserName` and `Environment.MachineName` to get these values at runtime



# Performing Authorization based on Windows Authentication

- Some variations

```
static void Main()
{
    AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal);
    WindowsPrincipal principal = (WindowsPrincipal)Thread.CurrentPrincipal;

    Console.WriteLine("My name is {0}", principal.Identity.Name);
    if (principal.IsInRole(WindowsBuiltInRole.Administrator)) {}
    if (principal.IsInRole(WindowsBuiltInRole.User)) {}
    if (principal.IsInRole(@"LEASTPRIVILEGE\Domain Admins")) {}
}

[PrincipalPermission(SecurityAction.Demand,
                    Role=@"LEASTPRIVILEGE\Domain Admins")]
static void ShutDownNuclearReactor() {}
```



# Performing Authorization by Using Application-Defined Roles

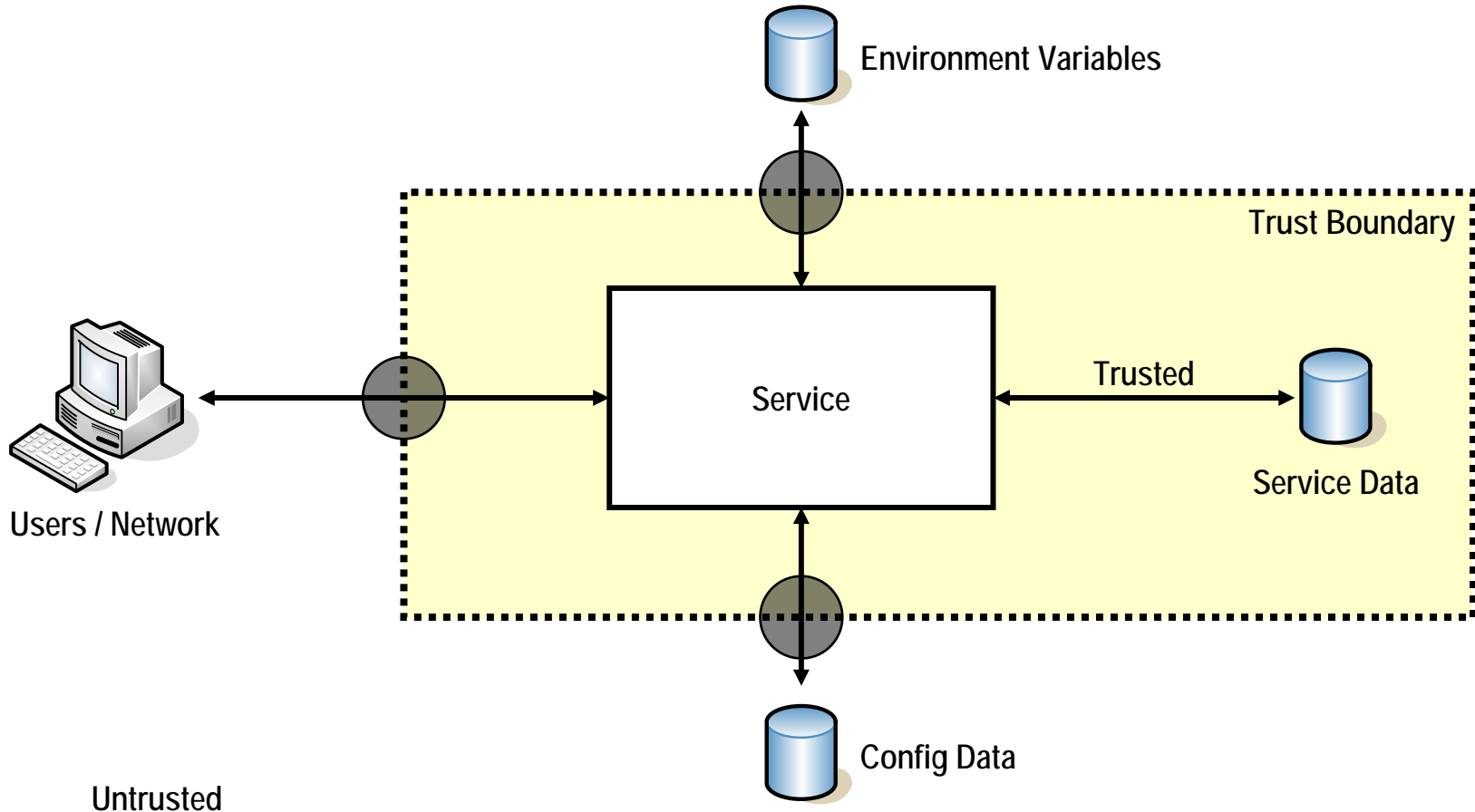
- To be independent from Windows Groups you can also use application defined roles
- Base the Identity on the Windows account and couple it with your own roles
- Base the Identity on a custom application login

```
GenericIdentity identity =  
    new GenericIdentity(WindowsIdentity.GetCurrent().Name);  
  
// normally you would look up the roles in a database or  
// config file  
Thread.CurrentPrincipal = new GenericPrincipal(identity,  
    new string[] { "Manager", "CanDeleteCustomer" } );
```



# Input Validation

- All Input is Evil!!!
- Create Trust Boundaries and Choke-Points



# How to validate Input?

- **Proactive vs Reactive**
  - Define allowed Input
  - The rest is forbidden
- **Regular Expressions are your friend**
- **Use Validation Frameworks**
  - ValidateRequest
  - ASP.NET Validator Controls
  - ExtenderProvider in WinForms



# Integrate with Windows Security

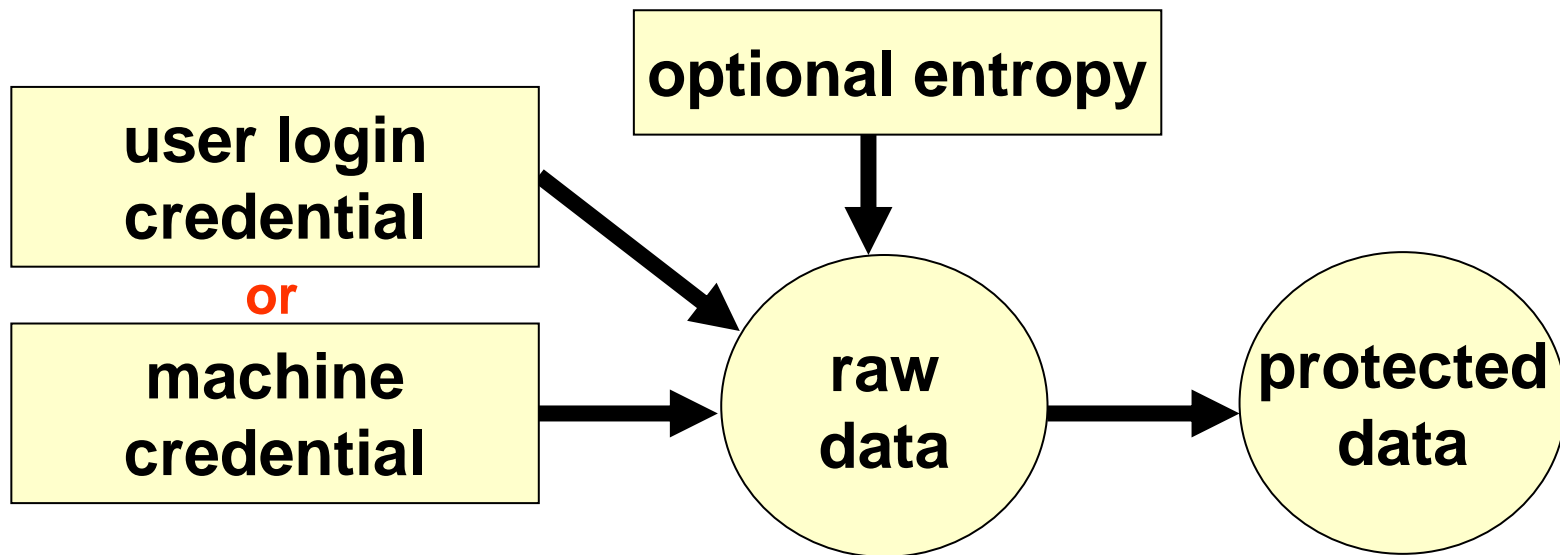
- **Don't reinvent the wheel**
- **Windows already provides**
  - user and group management
  - network authentication protocols
  - Impersonation / Delegation
- **Use**
  - ACLs
  - Logging / Auditing
  - DPAPI / Credential Manager / SSPI
- **New Windows Security Features at your disposal**
  - Kerberos constrained Delegation
  - Protocol Transition



# DPAPI

- **Data protection API**

- Encrypts and decrypts data based on the login credentials
- Supported on Windows 2000, XP, and 2003 server
- Can prompt the user for some entropy (e.g. password)
- The application can hard code the entropy
- Can use either the user or machine credentials



# Groups

- **There are four types of groups**
  - Some follow you all over the network (unlimited scope)
  - Others only show up in the scope where they are defined (machine or domain scope)

type of group	stored in	nest?	scope
<b>Universal</b>	active directory	yes	unlimited
<b>Global</b>	active directory	yes	unlimited
<b>Domain Local</b>	active directory	yes	single domain
<b>Local</b>	registry	no	single machine



# Code Access Security

- **CLR native Security Model**
- **Based on**
  - Evidence
  - Policy
  - Permission
- **Used in**
  - No Touch Deployment
  - ClickOnce
  - Partially Trusted ASP.NET
- **Useful for**
  - Deployment
  - Further refusing privileges

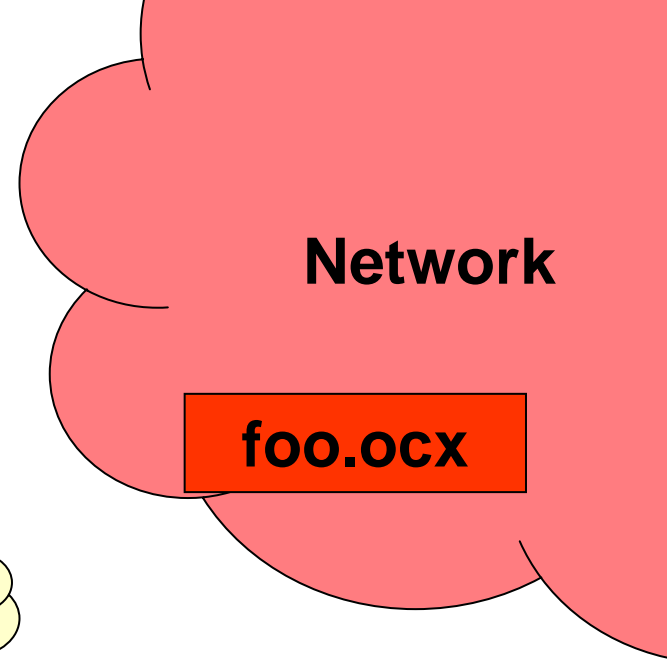
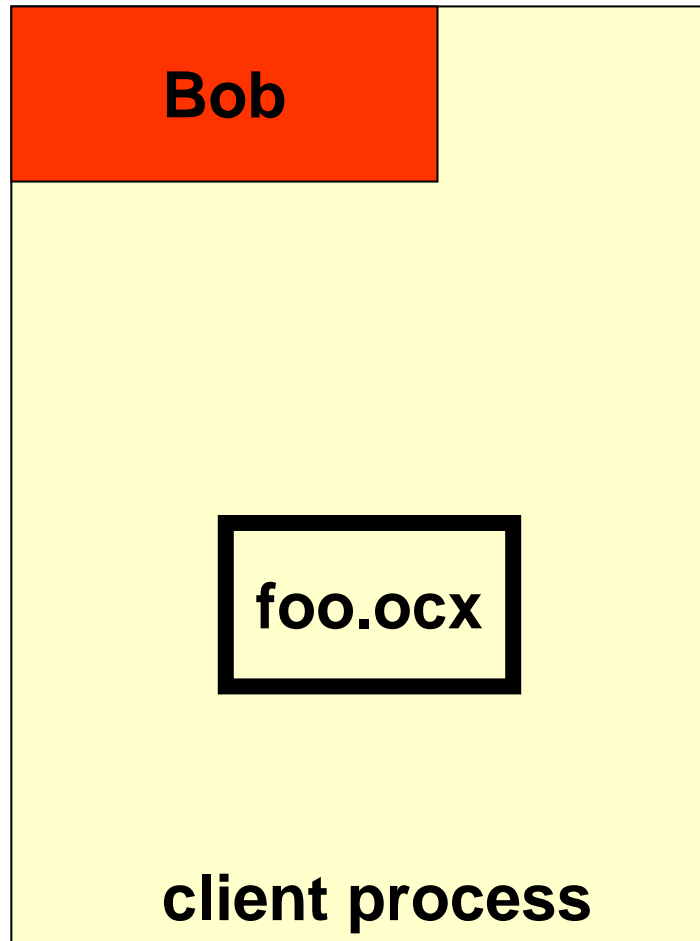


# The old model: Full Trust

- **In the COM era there were only two options for mobile code**
  - full trust
  - completely untrusted
- **User had to decide on trust level at download time**
  - user presented with certificate
  - asked whether they trusted the vendor and their code
- **NO = mobile code not allowed to execute**
  - usually means websites don't work
- **YES = mobile code allowed to execute**
  - might give richer browsing experience
  - could run more buggy and vulnerable code base
  - could install virus or Trojan horse



# The COM loader, illustrated



Uh, sure, if I have to...



Do you trust foo.ocx to do anything you can do?

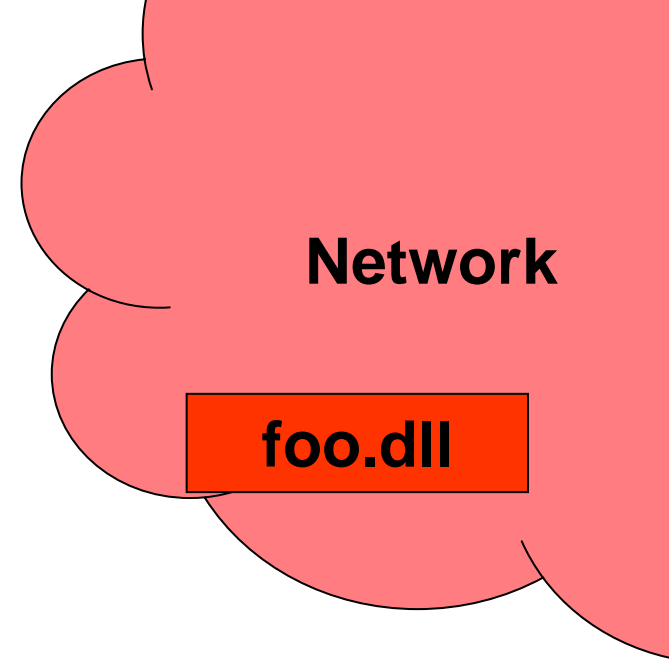
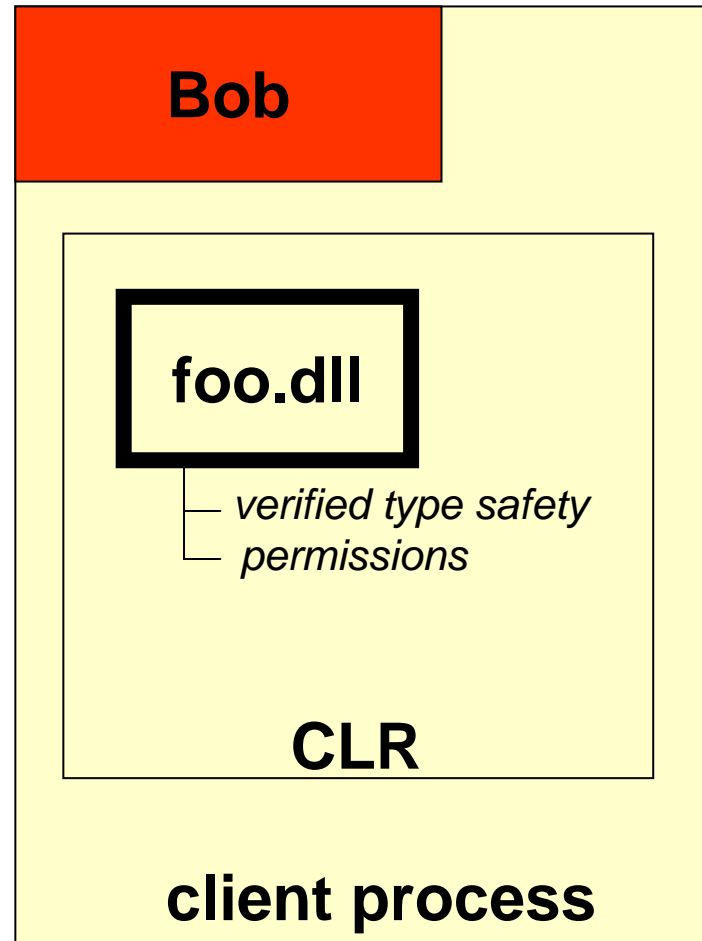


# The .NET assembly loader and trust levels

- **Each .NET assembly may run with different privileges**
  - explicitly installed code usually trusted
  - mobile code typically restricted
- **.NET uses Code Access Security (CAS) model**
  - assembly loader gathers **evidence** like source and public key
  - security **policy** used to evaluate evidence
  - assembly **permissions** determined from evidence and policy
  - CLR restricts assembly to actions allowed by its permissions
  - user never asked trust questions



# The .NET loader, illustrated



# The importance of type safety

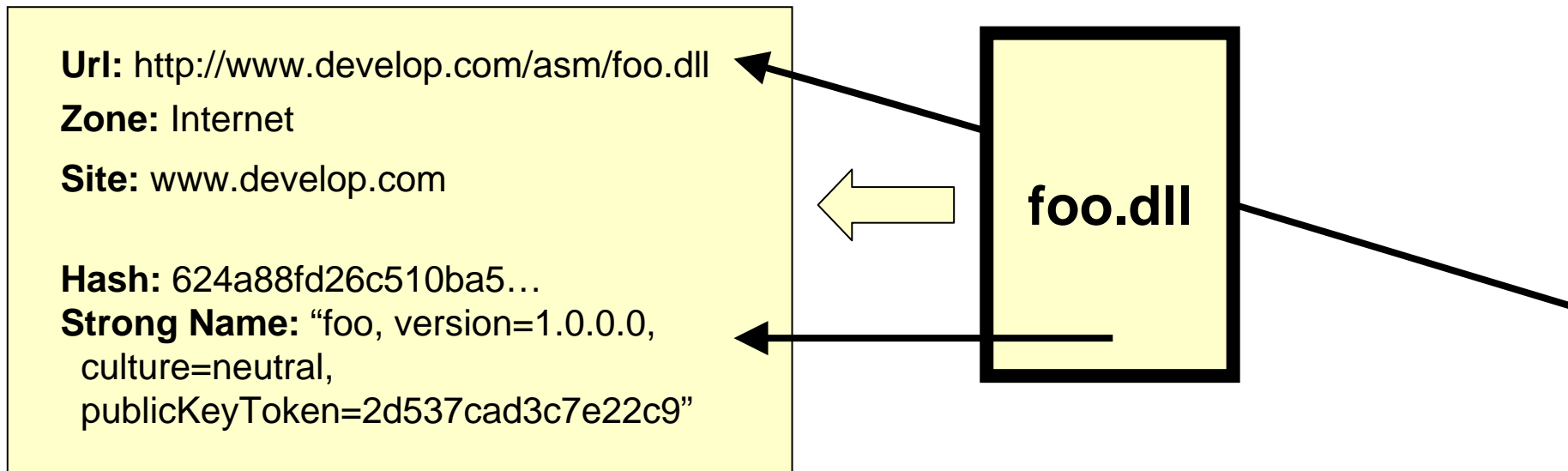
- **CLR enforces permissions to constrain mobile code**
  - possible because CLR type system is watertight
  - code is verified before execution to guarantee type safety
  - any bugs in the verification process compromises this system
- **COM based systems could not enforce type safety**
  - C++ allowed arbitrary casts
  - VB6 wasn't exempt either
  - no way to verify behavior of code



# Evidence

- **Evidence comes from properties of assembly**
  - source: url, zone, site, application directory
  - author: strong name, publisher (Authenticode)
  - contents: hash

## Evidence



# Evidence representation

- **FCL supplies classes used to represent evidence**
  - in `System.Security.Policy` namespace

```
Zone  
Url  
Site  
ApplicationDirectory  
StrongName  
Publisher  
Hash
```



# Permissions

- **Permissions limit what an assembly can do**
  - run if code not verifiable?
  - access file system?
  - access the network?
  - access certain environment variables?
  - call native code (COM objects, DLLs)?
  - access files or printers without asking user?



# CAS permission classes

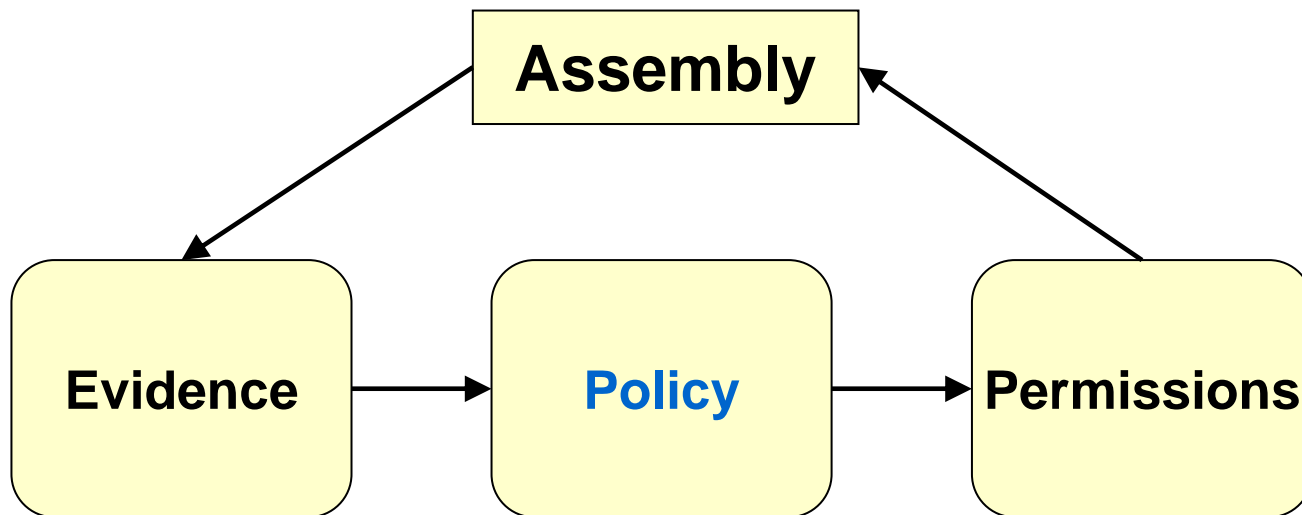
- **FCL supplies classes to represent permissions**
  - in `System.Security.Permissions` namespace

```
DBDataPermission
PrintingPermission
MessageQueuePermission
DnsPermission
SocketPermission
WebPermission
EnvironmentPermission
FileDialogPermission
FileIOPermission
IsolatedStorageFilePermission
ReflectionPermission
RegistryPermission
SecurityPermission
UIPermission
```



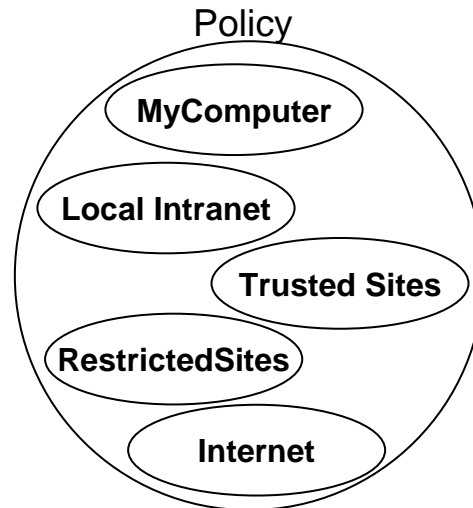
# Security policy

- **Security policy determines permissions granted to assembly**
  - evidence given to policy
  - policy grants permissions based on the evidence
  - resulting permissions assigned to assembly



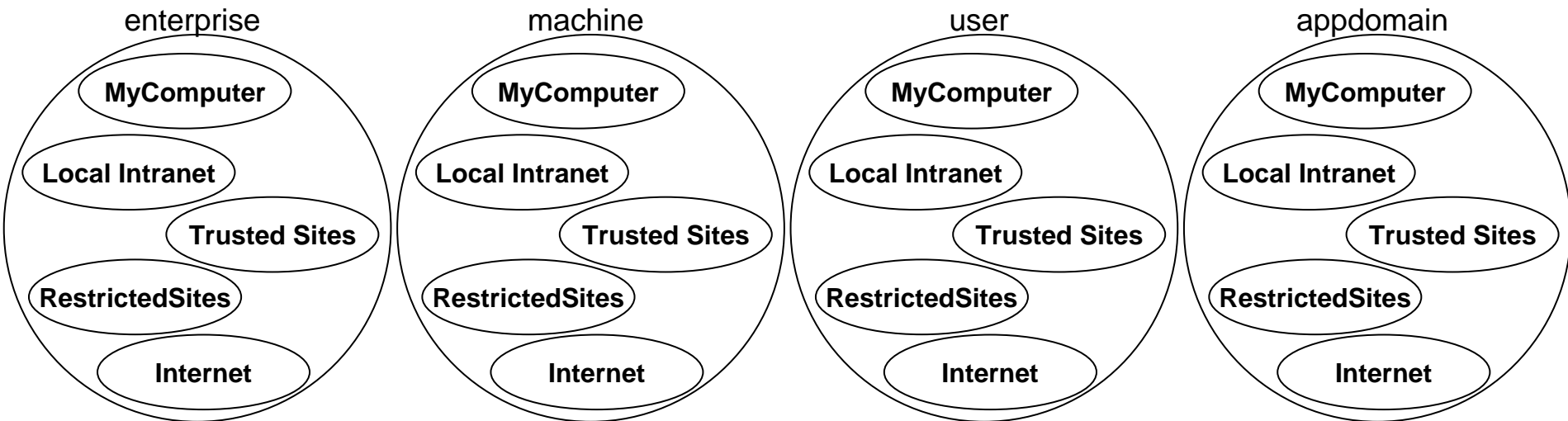
# Setting policy

- **By default, policy is defined based on Internet Explorer zones**
  - My Computer
  - Local Intranet
  - Trusted Sites
  - Restricted Sites
  - Internet



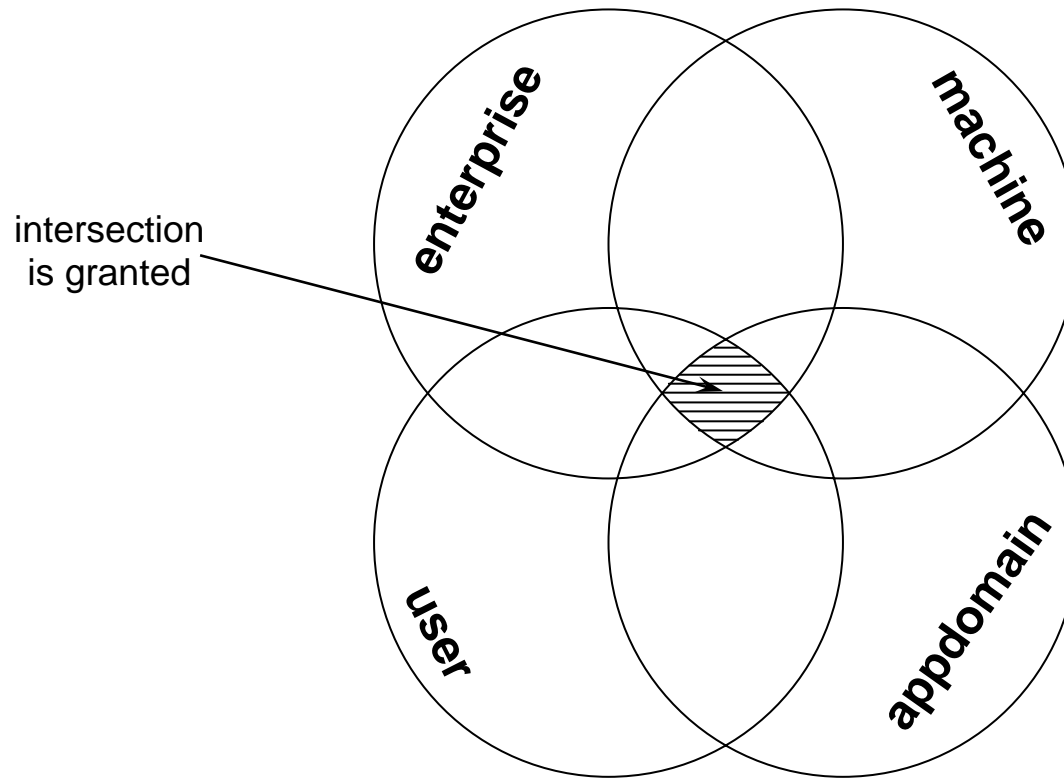
# Policy levels

- **Policy comes from four sources**
  - enterprise
  - machine
  - user
  - appdomain



# Policy interaction

- **Permission determined by intersecting applicable policies**
  - all policies must agree before a permission is granted



# Default permissions

- **Default permission set for each policy source**
  - enterprise, user, appdomain grant full trust to all code
  - machine has several restrictions

default machine policy

<b>My Computer</b>	<b>Full Trust (no limitations at all)</b>
<b>Local Intranet</b>	<b>Medium Trust</b>
<b>Trusted Sites</b>	<b>Low Trust</b>
<b>Internet</b>	<b>Low Trust</b>
<b>Restricted Sites</b>	<b>Nothing (cannot execute)</b>



# Customizing policy

- **Can customize each policy level**
- **Framework supplies tools to simplify task**
  - predefined permission sets
  - wizards to select settings
  - editors for finer-grained control
- **Framework supplies policy classes**
  - System.Security.Policy namespace
  - can alter policy programmatically for maximum control



# Full trust

- **No CAS restrictions whatsoever**
- **Code doesn't have to be verifiable**
  - all code is verified by default unless it requests SkipVerification
- **Can call out to native code without restriction**
  - DLLs
  - COM objects
- **Don't forget that the operating system still controls access to external resources based on tokens**
  - File system, kernel objects, etc.
  - DCOM
  - Databases



# Medium trust

- Code may execute.
- Code must be verifiable and may not call directly into unmanaged code.
- Allowed to assert permissions (more on this later).
- May not suspend, resume, interrupt, abort other threads, or stop its own threads from being aborted by more trusted code.
- May not modify security policy.
- May not change the managed principal on a thread.
- May not create or control AppDomains.
- May not serialize objects using a SerializationFormatter (allows access to private state).
- May not inject evidence for assemblies or AppDomains.
- May not configure System.Runtime.Remoting or add extensions such as remoting sinks.
- Denied access to all environment variables, except you may read USERNAME.
- May read or write to the local file system, but only by using open/save dialogs to get a stream (exception: may read directly from AppBase directory and any subdirectories)
- Unlimited isolated storage space for your assembly, scoped by user, assembly, and machine.
- May use reflection, but only to access public members of public types or to emit new types.
- May read and write from the clipboard and put up windows of any shape or size.
- May use DNS (Domain Name Service) without restriction.
- May submit print jobs directly to the default printer, or to any other printer using a common dialog.
- May read or write to existing event logs, and create event sources and logs.
- May not access the registry, SQL databases, message queues
- May use sockets, but only to connect to the site from which the code originated



# Low trust

- Same as Medium Trust, with the following further restrictions:
- May not assert permissions.
- May not read/write ANY environment variables.
- May not use the FileSave dialog to open files for writing (can still use FileOpen to read).
- May not read (via File IO) from AppBase directly without a dialog.
- Isolated storage further restricted by quota (10240 bytes), also must be scoped not only by assembly and user, but also by application.
- May only display “safe top-level windows” which is supposed to help keep mobile code from spoofing login dialogs and the like.
- While writing to the clipboard is unrestricted, only intrinsic controls such as the TextBox may read from the clipboard – user controls cannot.
- Cannot use reflection to emit code.
- Cannot use DNS (Domain Name Service) to resolve names.
- May not send print jobs directly to any printers. Must use the common print dialog in order to obtain the user’s consent.
- Only code from trusted sites can connect back to the original site



# No Trust

- No permissions are granted
- Code is not even allowed to execute



# Evaluating a policy level

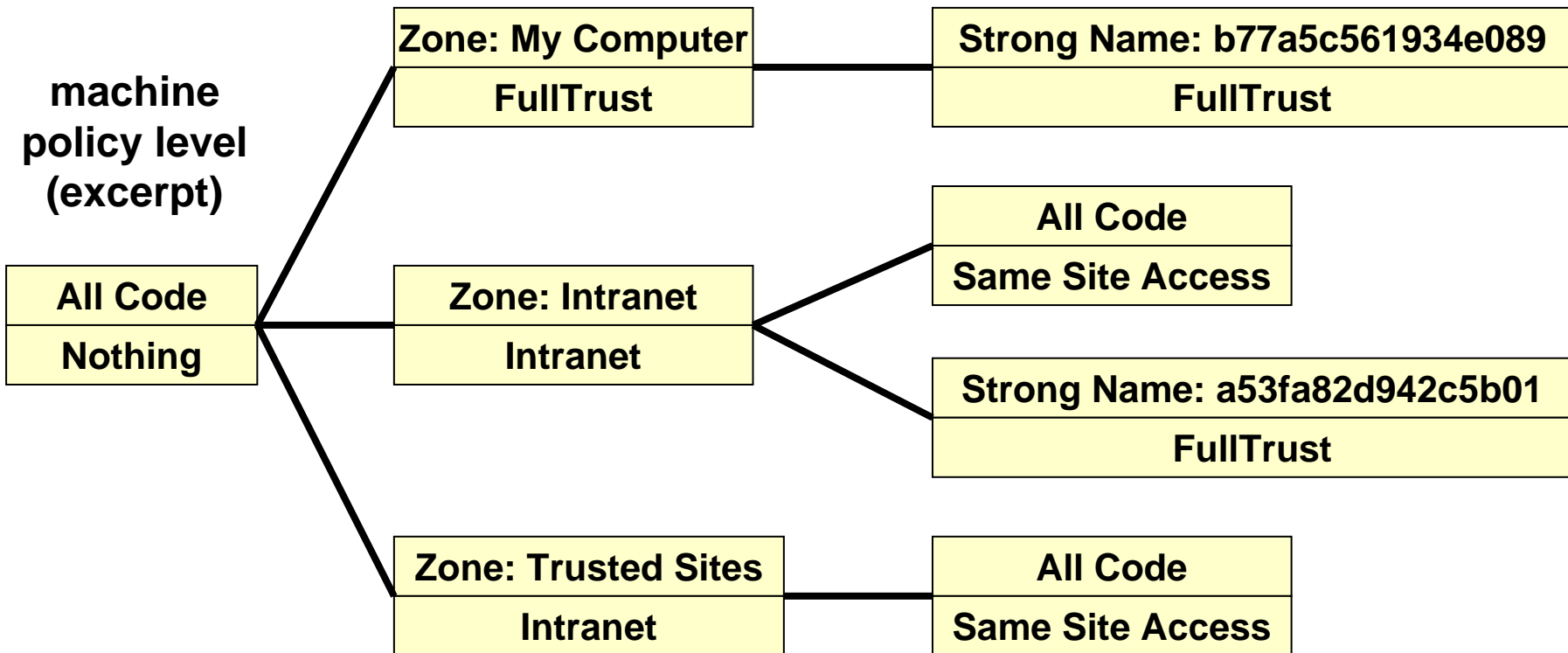
evidence:

<b>URL:</b> http://sales/routing.dll
<b>Zone:</b> Intranet
<b>StrongName:</b> a53fa82d942c5b01

permission grants:

<b>Nothing</b>
<b>Intranet</b>
<b>Same Site Access</b>
<b>FullTrust</b>

**= FullTrust**



# Instrumentation and Monitoring

- **We have to tell someone if something is going wrong**
- **Think Detection and Reaction**
  
- **Admins don't read log files**
  
- **We need 'easy-to-abstract' information**
  
- **Technologies**
  - Performance Monitor
  - Windows Management Instrumentation
  - Enterprise Instrumentation Framework
  - Logging Application Block



# Using Performance Monitor

- Easy to use
- Integrates in standard monitoring tools

```
PerformanceCounter total = new PerformanceCounter
    ("SuspiciousActivity", "ValidationErrorsTotal", false);

PerformanceCounter persecond = new PerformanceCounter
    ("SuspiciousActivity", "ValidationErrorsPerSecond", false);

...

total.IncrementBy(1);
persecond.Increment();
```



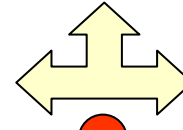
# WMI Architecture

Any application that understands automation

i.e. C/C++, VB, VBScript, Jscript, VBA, Perl

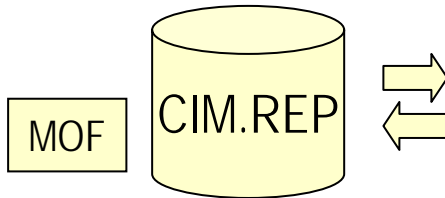
**Consumer**

Application



System.Management

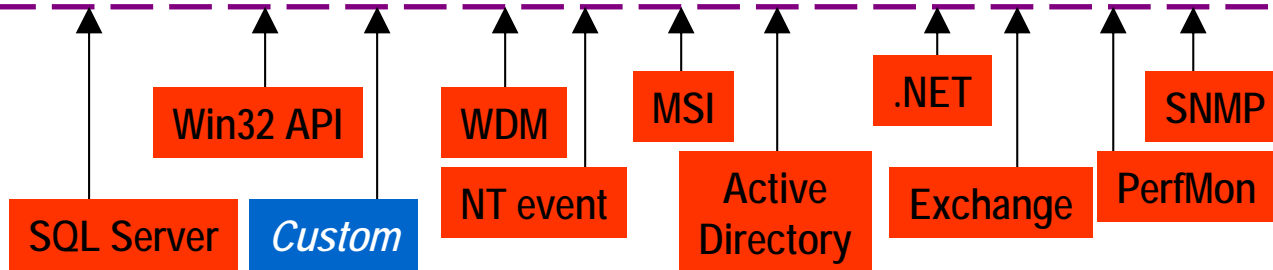
**CIM**



Windows Management Service  
%SystemRoot%\System32\WBEM\WinMgmt.exe

COM

**Provider**



# Creating WMI Classes

- **System.Management.Instrumentation provides a number of attributes that make it easy to instrument your application**
  - `[assembly: Instrumented("root/Leastprivilege")]`
- **Classes that should be published to the WMI repository**
  - are decorated with `[InstrumentationClass]`
  - or inherit from `Instance / BaseEvent`
- **Public fields and properties become WMI properties**
  - you can also apply the `[IgnoreMember]` Attribute
- **You have to register the class in the CIM**
  - `DefaultManagementInstaller` takes care of that
- **Support is limited to read properties**



# Creating WMI Classes

```
public class LogonFailureSummary : Instance
{
    public string Area;
    public int Count;
    public string LastFailedLogon;
}

[RunInstaller(true)]
public class MyProviderInstaller : DefaultManagementProjectInstaller
{
}

public class Program
{
    public static void Main()
    {
        LogonFailureSummary l = new LogonFailureSummary();
        l.Area = "Main App Logon";
        l.Count = 5; l.LastFailedLogon = DateTime.Now.ToString();

        l.Published = true; // publish the instance to WMI
    }
}
```



# Querying WMI Classes

```
LogonFailureSummary.LogonFailureSummaryCollection logons =  
    LogonFailureSummary.GetInstances();  
  
IEnumerator lenum = logons.GetEnumerator();  
  
while (lenum.MoveNext())  
{  
    LogonFailureSummary l = (LogonFailureSummary) lenum.Current;  
    Console.WriteLine("{0}: , {1}", l.Area, l.Count.ToString());  
}
```



# Creating WMI Events

```
public class SecurityEvent : BaseEvent
{
    public string Message;

    public SecurityEvent(string message)
    {
        this.Message = message;
        base.Fire();
    }
}

public class Program
{
    static void Main()
    {
        // logon procedure

        if (3 == numInvalidLogons)
            new SecurityEvent("Three invalid logons for user " + user);
    }
}
```



# Subscribing to WMI Events

```
WqlEventQuery eventQuery = new WqlEventQuery("SecurityEvent");
ManagementScope scope = new
    ManagementScope(@"\\.\root\LeastPrivilege\Demo");

eventWatcher = new ManagementEventWatcher(scope, eventQuery);
eventWatcher.EventArrived += new
    EventArrivedEventHandler(Delegate_EventArrived);

eventWatcher.Start();

private void Delegate_EventArrived(object sender, EventArrivedEventArgs e)
{
    updateEvents(Convert.ToString(e.NewEvent.Properties["Message"].Value));
}
```



# Security on each Layer

Secure Communication

## User Interface

- How do I customize the user interface based on a user's role?
- How do I validate user input?
- Does the Front-End run with Least Privilege / Partial Trust?
- How do I store secrets?

## Business Logic

- How can I write code to manage authorization in business processes?
- How do I make sure I know what identity to authorize on in server environments?
- How should I build reaction and detection in the Business Tier
- Does the Business Tier run with Least Privilege / Partial Trust?

## Data Access

- How do I prevent sensitive data from reaching unauthorized users?
- How do I ensure anonymous users don't modify data?
- How can I store sensitive data?



# User Interface

- **Application must work with non-admin privileges**
  - Test with Least Privilege
  - Write with Least Privilege
  - Always store data in the user profile
- **Decide which role based approach to take**
  - Windows roles
  - Custom roles
- **Is Partial Trust an option for you?**
  - Interesting deployment scenarios
  - Be careful with file operations, consider IsolatedStorage
  - Some stuff simply doesn't work, e.g.
    - Remoting
    - Web Service Enhancements (WSE)



# Altering the User Interface

```
// Alter Controls
if (User.IsInRole("CanApproveClaims")) {
    cmdApprove.Visible = true;
    ClaimGrid.Columns[3].Visible = true;
}
```

```
// Alter Page Flow
if (User.IsInRole("GeneralManager"))
    Server.Transfer("ChooseDepartment.aspx");
else if (User.IsInRole("StoreManager"))
    Server.Transfer("ChooseEmployee");
```

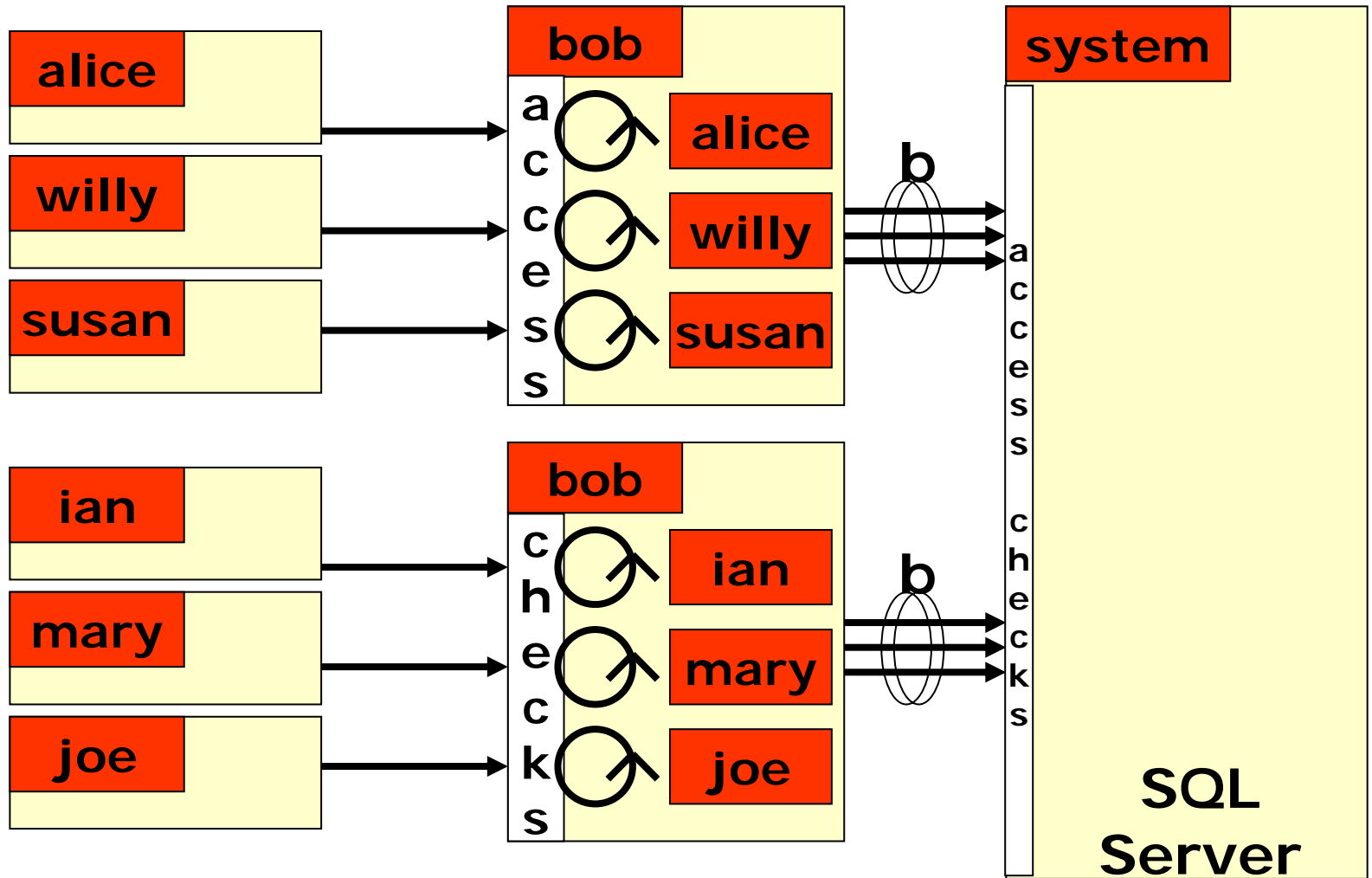


# Security in the Business Tier

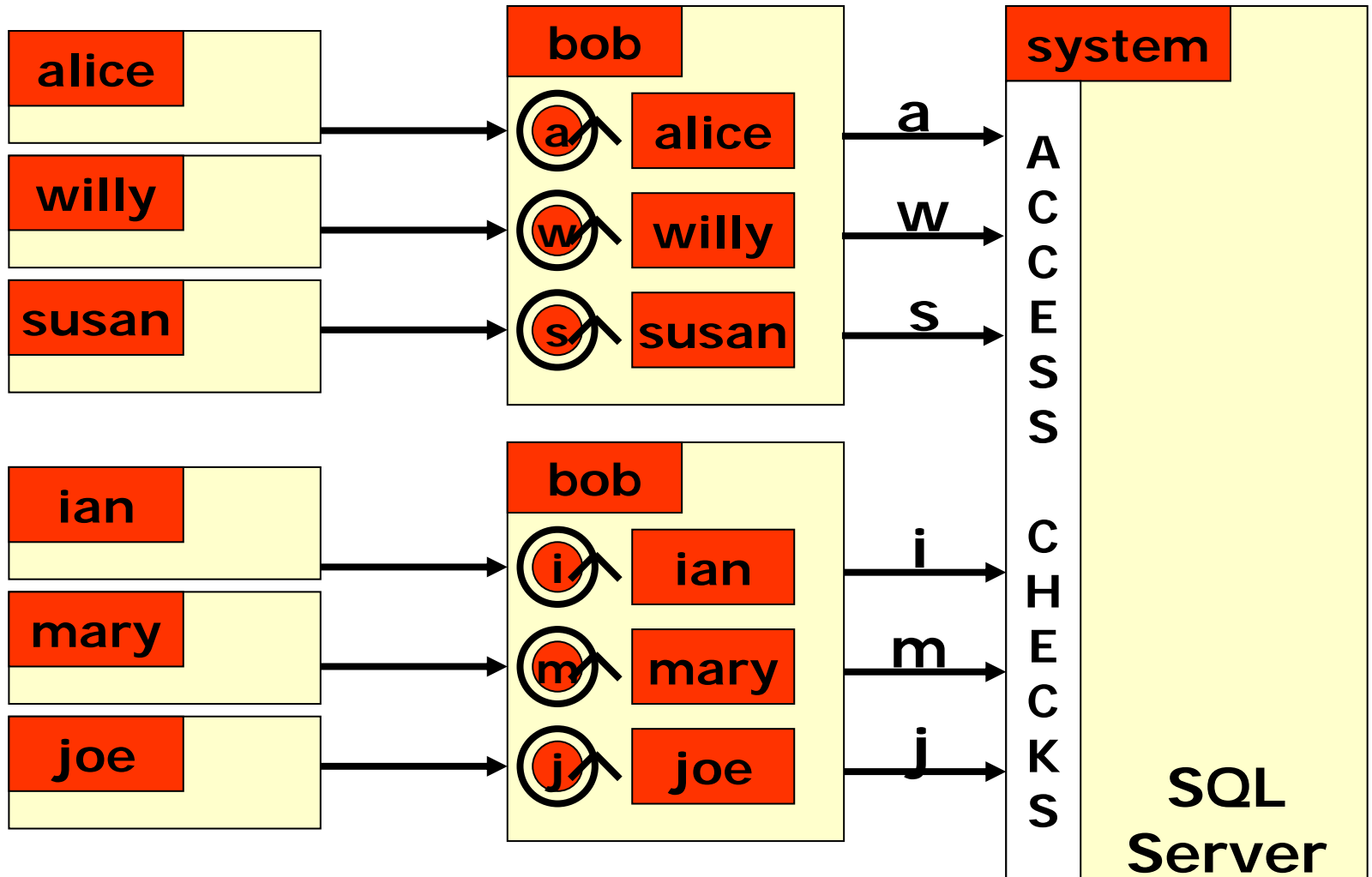
- **Authorize an operation that performs a business process**
- **Authorize calls to other parts of your distributed system or external services that are not part of your application**
  
- **Decide which Business Tier technology you want to use**
  - ASP.NET / WSE
  - COM+
  - .NET Remoting
  
- **Decide which security model you want to use**
  - Impersonation/Delegation Model
  - Trusted Subsystem Design



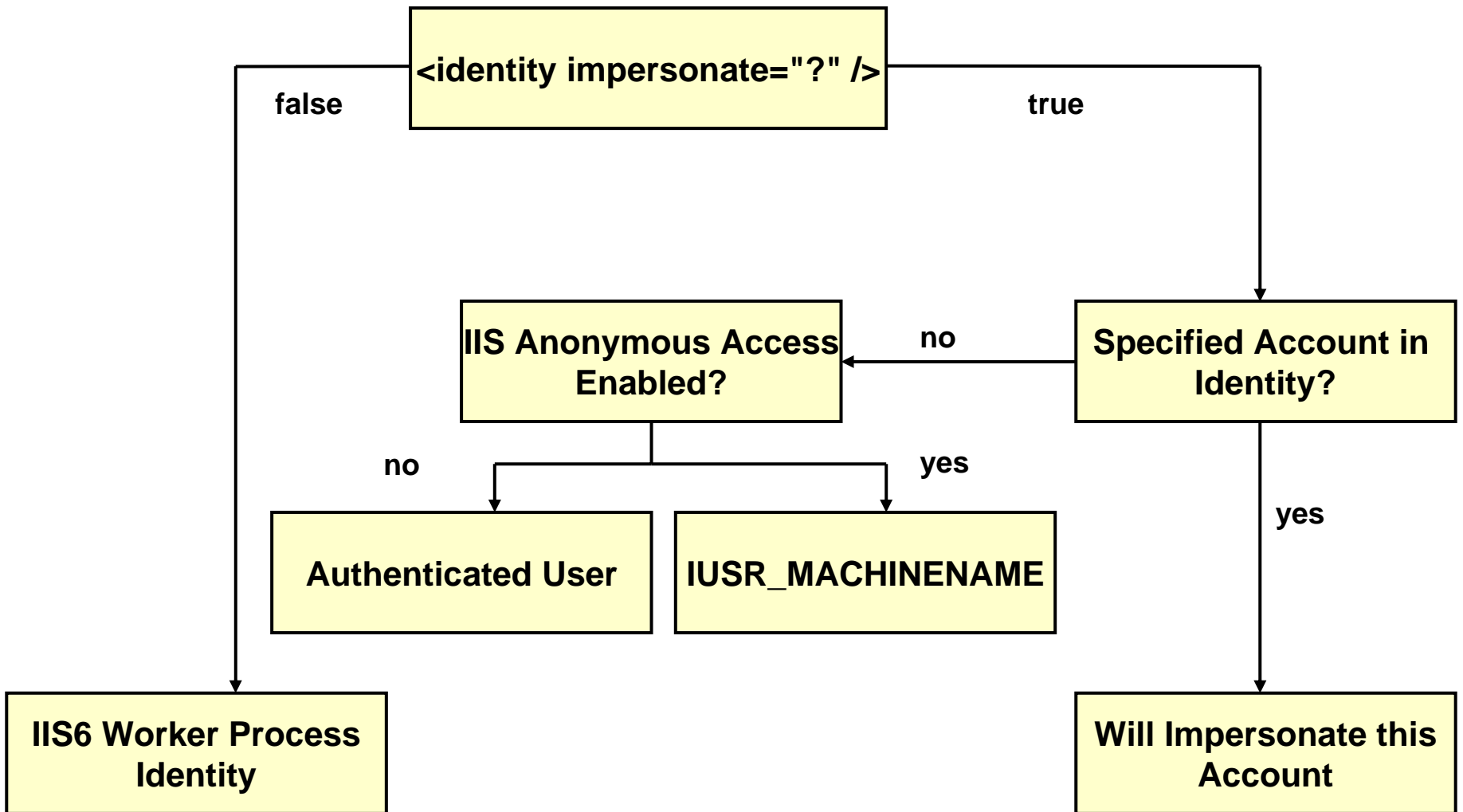
# Trusted Subsystem



# Impersonation/Delegation



# ASP.NET and Impersonation



# Programming Impersonation

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

[PrincipalPermission(SecurityAction.Demand, Authenticated=true)]
void DoSomeWorkForMyClient() {

    WindowsIdentity client = (WindowsIdentity)Thread.CurrentPrincipal.Identity;

    // temporarily impersonate the client to access a file
    WindowsImpersonationContext ctx = client.Impersonate();
    try {
        // open a file on the client's behalf
        using (Stream s = new FileStream("someFile.txt", FileMode.Open)) {
            // use the client's file...
        }
    }
    finally {
        ctx.Undo();
    }
}
```



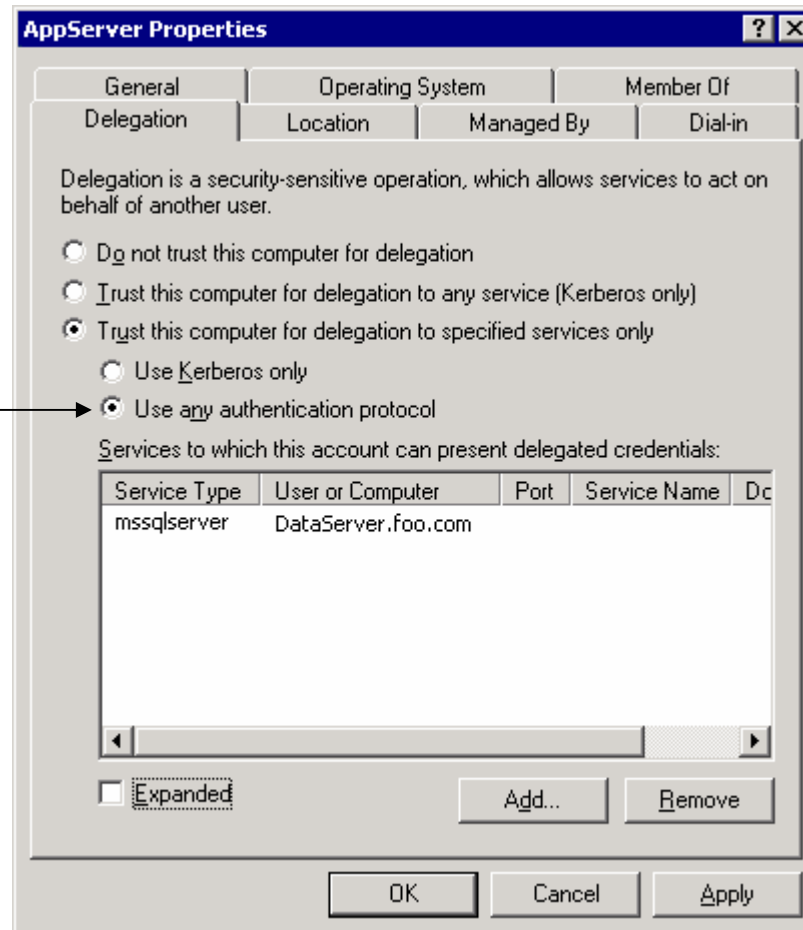
# Windows Server 2003 protocol transition

- **Protocol transition allows trusted servers to obtain tickets on a client's behalf**
  - without knowing the client's password
- **Tickets can be translated into tokens**
  - token can be used to discover groups
  - token can be impersonated to access local resources (service must be granted `SeImpersonatePrivilege` to do this)
- **Token may also be used to access network resources**
  - via constrained delegation
  - only possible if “allowed-to-delegate-to” list is populated
- **Thus the term “protocol transition”**
  - Server is trusted to transition from one authentication protocol (used to authenticate the client) into Kerberos on the back end

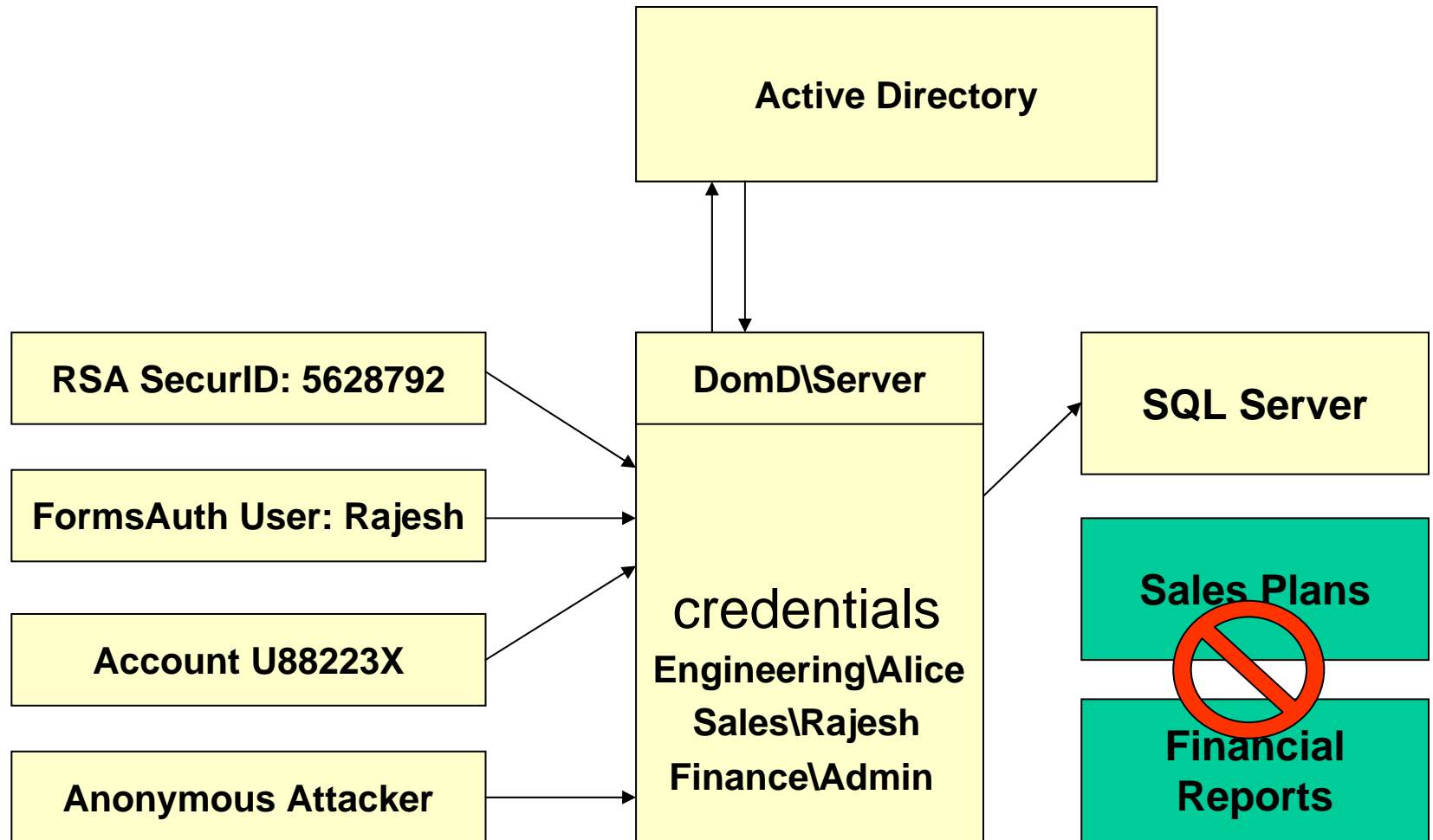


# Configuring protocol transition

“Protocol transition”



# Protocol transition, illustrated



# Partially Trusted ASP.NET

- **Full Trust is quite powerful**
  - call out to unmanaged code
  - start processes
  - read other web sites' data
  - use reflection
  
  - ...can even grab security tokens from the process space and impersonate them
- **Very important if you share the application server with other parties**



# Using Code Access Security in ASP.NET

- machine.config or web.config

```
<location allowOverride="true"> <!--allow web config to override
<system.web>
  <securityPolicy>
    <trustLevel name="Full" policyFile="internal"/>
    <trustLevel name="High" policyFile="web_hightrust.config"/>
    <trustLevel name="Medium" policyFile="web_mediumtrust.config"/>
    <trustLevel name="Low" policyFile="web_lowtrust.config"/>
    <trustLevel name="Minimal" policyFile="web_minimaltrust.config"/>
  </securityPolicy>
  <trust level="Full" originUrl=""/>
</system.web>
</location>
```



# Partitioning Business Logic

- **Your service demons should always run least privilege**
- **Sometimes you need to perform privileged work**
- **Keep the privileged code out of your demon**
  
- **For Windows Security**
  - Do a code review
  - Put the code in an Enterprise Services component
  - Give that ES component elevated privileges
  - Be sure to tightly ACL that component
  
- **For CAS**
  - Do a code review
  - Put the code in the GAC
  - Apply `[AllowPartiallyTrustedCallers]`



# Security in the Data Tier

- Prevent sensitive data from propagating outside the data tier
- Prevent unauthorized modification of data
- Always limit access to data by using stored procedures or views rather than granting direct access to tables
- Use the security features of your DBMS to limit access
- Never store sensitive data in clear text
- Use salted hashes for passwords
- Consider using different connection strings / accounts for different parts of your application
  
- SQL Server transmits data in clear text by default
  - use SSL or IPSEC



# Storing sensitive Data

- **You don't want clear text passwords or credit card numbers on your server**
- **Use encryption**
  - symmetric cryptography
  - public/private key cryptography
- **For passwords use salted hashes**
  - pick a random number (=salt)
  - pick a hash algorithm, e.g. SHA-1
  - $SH = H(\text{salt}, H(\text{password}))$
  - repeat several times
  - ..or use `PasswordDeriveBytes`



# Performing Authorization in the Data Tier

- Removing data based on role membership

```
public DataSet GetEmployeeData()
{
    // retrieve data
    if (!Thread.CurrentPrincipal.IsInRole("Manager"))
        data.Tables["HR"].Columns.Remove["Salary"];

    return data;
}
```



# Performing Authorization in the Data Tier

- ...or perform different work in stored procedures

```
create procedure GetEmployeeData
  @EmployeeName varchar(50),
  @IsManager bit
as
  if @IsManager = 1
    select * from Employees where EmployeeName = @EmployeeName
  else
    select FirstName, LastName from Employees
```



# Performing Authorization in the Data Tier

- **Column based security is easy in databases**
  - Use the built in System Authorization
- **Row based security is harder**
  - When using impersonation the callers identity flows to SQL Server and is available via `SUSER_SNAME`
  - Can get complex with more advanced scenarios

```
create view EmployeeData
as
  select * from Employees where EmployeeName = SUSER_SNAME()
```



# App Servers and Communication Protocols

- **ASP.NET (ASPX / ASMX)**
  - is tied to HTTP
  - always use SSL
  - no 'callbacks'
  - rich process & programming model
- **ASP.NET (ASMX + WSE)**
  - WS:Security provides rich security APIs
  - Currently very 'close to the metal'
- **WSE**
  - You have to manage processes yourself
  - Listener can be implemented in arbitrary apps, e.g. WinForms



# App Servers and Communication Protocols

- **COM+ (and MSMQ)**
  - tightly integrated with Windows Security
  - supports confidentiality, integrity, authenticity
  - 'drag & drop' security
  - rich process & programming model + services
- **Remoting**
  - Only supports security when hosted in IIS
  - Callbacks are never authenticated/encrypted
  - .NET 2.0 supports Kerberos and IPC



# Summary

- **Security is a feature – design it as such**
- **Do threat modeling first**
  
- **Follow best practices**
  - input validation
  - least privilege
  
- **100% security is impossible**
- **Build detection and reaction in your apps**
- **Try to integrate with your environment as much as possible**
  
- **Do (third party) audits or penetration tests**

