Table 4.1: Resource Access Decision Matrix *(continued)*

Security Consideration	Impersonation and Constrained Delegation	Trusted Subsystem
A client is authenticated by sending the user name and password of the client in the Web service message.	The user name and password can be used to authenticate the client with Windows authentication to support both impersonation and constrained delegation.	Can be implemented using a trusted business identity or the service account to authenticate with the resource.
The client's identity must be passed to resources.	When using impersonation or constrained delegation, the client's identity is passed using operating system capabilities to downstream resources.	The client's identity would need to be passed as part of the message header or body.
Remote resources need to access another resource using the original client's credentials.	Constrained delegation must be used to flow the client's credentials to a remote resource.	Not applicable.
Resources need to perform actions based on the identity of the client.	If only the identity is required, impersonation can be used; otherwise, constrained delegation must be used.	If only the identity is required, it can be passed as part of the message header or body; otherwise, this is not an option.

The remainder of this chapter focuses on the following design pattern and technical supplement:

- Trusted Subsystem
- Protocol Transition with Constrained Delegation Technical Supplement

Trusted Subsystem

Context

A client needs to access one or more Web services that are distributed across a network. The Web services are designed so that access to additional resources (such as databases or other Web services) is encapsulated in the business logic of the Web service. These resources must be protected against unauthorized access.

Problem

How do you ensure that the client that is used to access the Web service cannot access the additional resources directly?

Forces

Any of the following conditions justifies using the solution described in this pattern:

- **Security policy prohibits users from accessing downstream resources directly**. Direct access to remote resources such as a database or Web services may result in business logic being circumvented and cause data inconsistencies in underlying data stores.
- **Remote resources cannot validate user credentials**. The downstream resources may exist in a security domain that is different from the one where the client was authenticated, or the authentication protocol that was used to authenticate the client may not support delegation to the remote resource.
- **There is a risk that resources can be exploited if the Web service is compromised by an attacker**. The surface area for attackers can be reduced by restricting access to a small group of accounts. This can also simplify management of access rights for the resource.

The following condition is an additional reason to use the solution:

- **The application or Web service can take advantage of resource sharing optimization techniques**. Resource sharing optimization techniques may include connection pooling and caching.

The following condition is not resolved by the base pattern, but is resolved by Extension 1 — Flowing the Identity of the Client:

- **Resources need to perform actions based on the identity of the client**. For example, actions performed in a database may require a client identity to support data entitlement logic or to create an audit trail.

For more information, see the "Extensions" section at the end of this pattern.

Solution

The Web service acts as a trusted subsystem to access additional resources. It uses its own credentials instead of the user's credentials to access the resource. The Web service must perform appropriate authentication and authorization of all requests that enter the subsystem. Remote resources should also be able to verify that the midstream caller is a trusted subsystem and not an upstream user of the application that is trying to bypass access to the trusted subsystem.

Participants

The Trusted Subsystem pattern involves the following participants:

- **Client**. The client accesses the trusted subsystem and provides the credentials for authentication during the request to the trusted subsystem.
- **Trusted Subsystem**. A Web service that accesses the downstream resource and replaces the client's security context with its own.
- **Remote Resource**. A Web service, database or other major component of a system. Access to the remote resource is controlled to prevent unauthorized use.

Process

Figure 4.1 depicts the interactions performed when a downstream resource is accessed through a trusted subsystem.

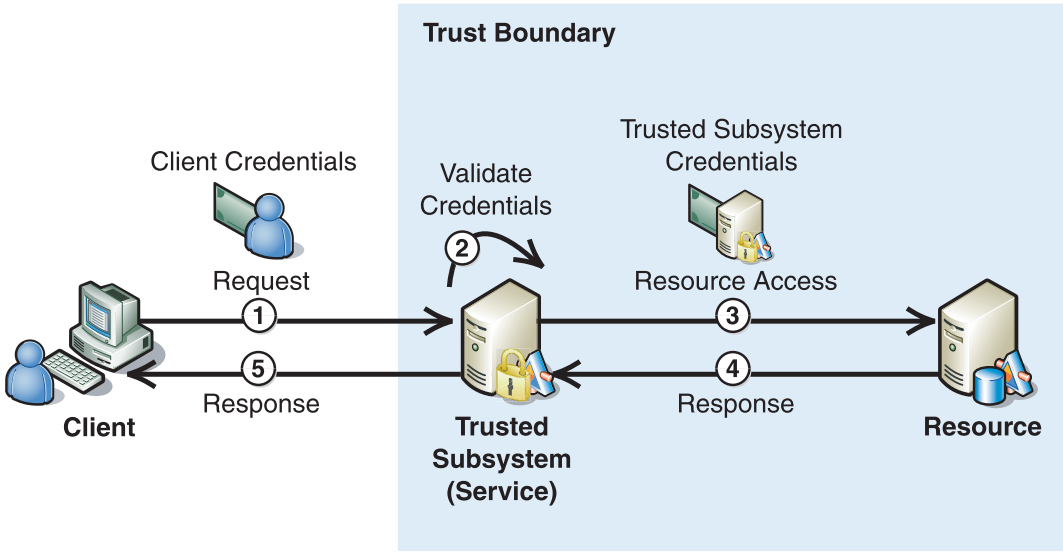

Figure 4.1
Trusted subsystem

As illustrated in Figure 4.1, the trusted subsystem process is described in the following steps:

1. The client submits a request to the trusted subsystem. The client provides credentials to the trusted subsystem.
2. The trusted subsystem authenticates and authorizes the user. Authentication can be direct or brokered. For more information, see the Direct Authentication pattern and the Brokered Authentication pattern in Chapter 1, "Authentication Patterns."
3. The trusted subsystem sends a request message to the remote resource. This request is accompanied by the credentials for the trusted subsystem (or the service account under which the trusted subsystem process is being executed).
4. The downstream resource authenticates and authorizes the trusted subsystem. It then processes the request and issues a response to the trusted subsystem.
5. The trusted subsystem processes the response and issues its own response to the client.

When multiple Web services collaborate to solve more complex problems, a Web service can simultaneously be a trusted subsystem and a resource that is accessed by a trusted subsystem. Figure 4.2 shows two overlapping trusted boundaries, with the trusted subsystem 1 taking responsibility for authenticating the client and the trusted subsystem 2 taking responsibility for authenticating trusted subsystem 1. Trusted subsystem 2's credentials are then used to access the remote resource.

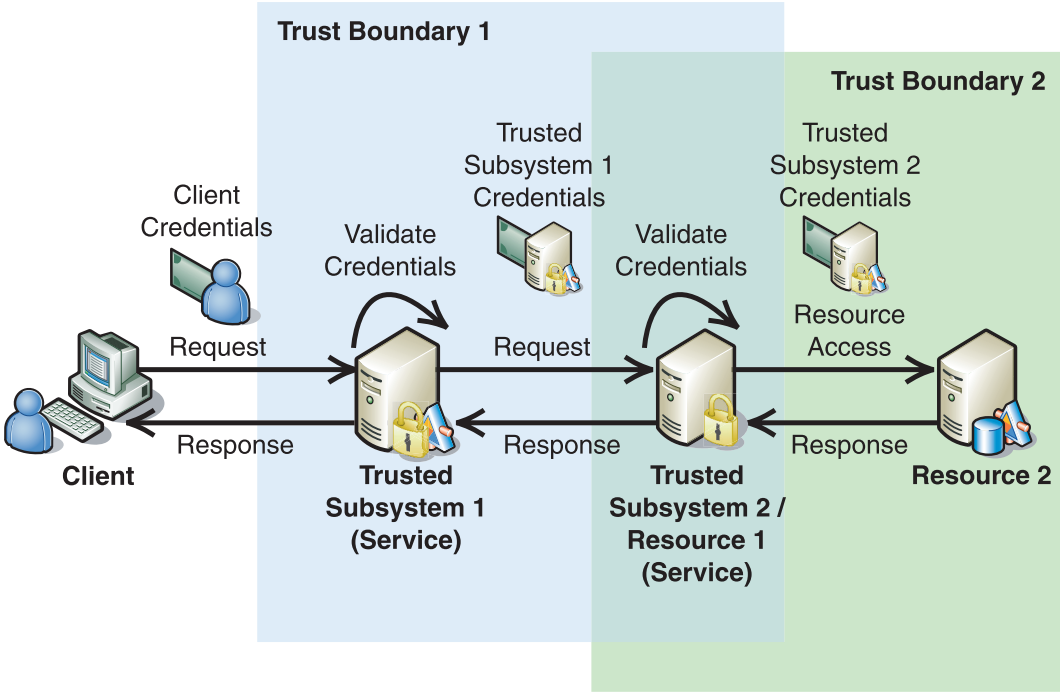

Figure 4.2
A Web service acting as a trusted subsystem and also as the resources of a trusted subsystem

Enforcing the Trust Relationship

Downstream resources must be able to verify that the midstream caller is a trusted subsystem and not just any system process. Requiring this type of verification enhances security by making it more difficult for attackers to simulate a trusted subsystem and perform man-in-the-middle attacks. Several approaches can be used to implement trusted subsystem verification:

- Authenticate the trusted subsystem with a Kerberos protocol service account.
- Use local accounts on each host.
- Use an X.509 PKI for authentication within the trusted subsystem.
- Secure communications by using IPSec between the computers in the trusted subsystem.

Kerberos Protocol Service Accounts

A common approach to implement verification with the Kerberos protocol is to use a service account that is used only within a particular trusted subsystem. This approach requires the service to be authorized so that only the trusted subsystem account can access it.

Local Accounts

When it is not possible to authenticate with a Kerberos protocol Key Distribution Center (KDC) you can create a local account on each host within the trusted subsystem. Each account has the same login and password. Accounts that are created to function this way are often referred to as mirrored accounts. While this approach provides a simple solution, it should not be your first choice. If you chose to use mirrored accounts, you should ensure that you use complex passwords and change them frequently.

X.509 PKI

An X.509 PKI can issue a certificate for each application within the trusted subsystem. The control of access to resources within the trusted subsystem is based on the ability of an application to prove possession of the certificate private key. It does this in conjunction with validating the certificate against a list of certificates that are authorized to access the resource.

IPSec

IPSec secures messages between two hosts at the network layer to provide data confidentiality, data integrity and replay detection. It can be configured to initiate secure communications with the Kerberos protocol, X.509 certificates, or a pre-shared key. IPSec performs considerably better than message layer security, but it does not allow for granular control of resources. This is because a trusted subsystem, which is established with IPSec, can only be established between the computers that participate in the trusted subsystem, and not based on a specific application accessing a specific resource.

Example

Global Bank provides a customer account client application that accesses a centralized account management database through a Web service. The client application must authenticate with the Web service to use the account management database.

In this scenario, the Web service acts as a trusted subsystem by using its own credentials to access the account management database. The client application cannot directly log in to the account management database because this violates the security policy and bypasses the business logic.

Note: This example usually requires data entitlement logic to ensure that after a customer has authenticated, he or she cannot access account details for another account. For more information, see "Extension 1 — Flowing the Identity of the Client" at the end of this pattern.

Resulting Context

This section describes some of the more significant benefits, liabilities, and security considerations of this pattern.

Note: The information in this section is not intended to be comprehensive. However, it does discuss many of the issues that are most commonly encountered for this pattern.

Benefits

The benefits of the Trusted Subsystem pattern include the following:

- Access to the downstream resource is simplified, which allows you to take advantage of optimizations such as connection pooling to improve performance.
- Administration of access control lists on downstream resources can be simplified because only the trusted subsystem is allowed access to the resources.
- The attack surface of the Web service is reduced by limiting the resources that are authorized to access it directly.

Liabilities

If a trusted subsystem is compromised, the trusted subsystem can be used to exploit the downstream resource, potentially on behalf of a legitimate user. For this reason, trusted subsystems are often a choice target for attackers to probe for vulnerabilities. Care must be taken to ensure that a trusted subsystem is very secure.

Security Considerations

A downstream system should be able to verify that the caller is a trusted subsystem and is not an authenticated system process. This can be accomplished by establishing security claims that are issued by a trusted third party and that are verified by the downstream resource.

Extensions

The extension described here builds on the base pattern to provide additional capabilities. In addition to resolving the forces stated for the base pattern, this extension also resolves the following condition:

- **Resources need to perform actions that are based on the identity of the client**. For example, actions performed in a database may require a client identity to support data entitlement logic or to create an audit trail.

Extension 1 — Flowing the Identity of the Client

In a trusted subsystem model, the credentials of the originating client are not used for authentication purposes against downstream resources. However, in many cases, the resource must perform authorization or data entitlement checks that are based on the identity of the originating client — and not on the identity of the trusted subsystem.

You can use the following two main approaches for flowing an identity:

- The trusted subsystem provides a self-signed token.
- The trusted subsystem forwards a signed token that is provided by an authentication broker.

Note: As with any data that is passed from a trusted subsystem to a downstream resource, the downstream resource relies on the integrity of the trusted subsystem. In each approach described here, the client's identity is simply flowed as part of the message from the trusted subsystem. It is not possible to detect if the trusted subsystem substituted one user's signed or unsigned credentials in place of alternative (perhaps cached) credentials for malicious reasons.

Approach 1 — The Trusted Subsystem Provides a Self-Signed Token

With this approach, the identity that the trusted subsystem sends to the resource is included in the message. You can include the identity in the message the following two ways:

- Include the client's identity as metadata in the SOAP message header. This can be performed by using a custom SOAP header or by using a WS-Security **UsernameToken** without a password.
- Include the client's identity in the body of the message.

In both cases, the message (including the client's identity) must be signed by the trusted subsystem, so that the downstream resource can authenticate the trusted subsystem and perform data origin authentication. The resource must assume that the trusted subsystem has authenticated the client whose identity is contained in the message. Otherwise, it has no way to know directly that the client has been authenticated.

Approach 2 — The Trusted Subsystem Forwards a Signed Token That Is Provided by a Trusted Third Party

With this approach, the trusted subsystem is responsible for forwarding a token to the downstream resource that is signed by a trusted third party, such as a Security Token Service (STS). The downstream resource can then validate the client's claims within the token, based on the trust relationship with the STS. It also allows the resource to verify that the client was authenticated recently. For this reason, the tokens issued by the STS should have a short lifetime.

When the client authenticates with the STS, the STS issues a signed security token that contains claims, such as the client's identity and roles. The token is used by the client to authenticate with the trusted subsystem. After the trusted subsystem receives the security token and authenticates the client, it signs the token and forwards it in a signed message to the downstream resource. The downstream resource authenticates the trusted subsystem and it is also able to verify the clients token using the signature of the STS within the forwarded token.

Protocol Transition with Constrained Delegation Technical Supplement

Consider the following scenario:

> *You are deploying a Web service that does not use Windows integrated authentication. After a client is authenticated; the client needs to be transitioned to a Windows account so that role-based authorization can be performed. The Web service also needs to interact with Web services or other downstream resources that can only be accessed with valid Windows credentials.*

The common approach to this problem is to have the client application send a user ID and password that can be used for authentication within the Web service. However, this requires the client application to store a password for use when it accesses the Web service. In addition, the password needs to be protected while it is in transit between the application and the Web service. Both of these requirements represent a security risk that should be avoided.

The solution to this problem is to use the new Kerberos protocol extensions in Windows Server 2003. The new extensions require the user ID but not the password. You still need to establish trust between the client application and the Web service; however, the application is not required to store or send passwords. One of these extensions, referred to as Protocol Transition, can initialize a valid **WindowsIdentity** object with only the user ID. The other extension uses the new **WindowsIdentity** object with constrained delegation to access remote resources.

The new extensions are:

- The Kerberos protocol transition extension, S4U2Self.
- The Kerberos constrained delegation extension, S4U2Proxy.

The following list identifies three distinct operations that you can implement with the new extensions:

- Use protocol transition to initialize a **WindowsIdentity** object for authorization checks.
- Use protocol transition to initialize a **WindowsIdentity** object for impersonation.
- Use constrained delegation to access remote resources.