WINDOWS IDENTITY FLOW & AZURE SERVICE BUS QUEUES

Abstract

This paper is a walk through demonstrating how you could flow a windows identity through Windows Azure Service Bus and use constrained delegation to access downstream resources.

Michael Stephenson

About the Author

Michael Stephenson

Michael Stephenson is a UK based integration and cloud specialist currently working for Connected Systems Consulting and who has worked with many consultancies and customers building hybrid integration solutions using Microsoft technologies such as BizTalk & Windows Azure. Michael is also a Pluralsight trainer and blogger on integration and architecture related topics and was the creator of the BizTalk Maturity Assessment. Michael has been a Microsoft MVP for 6 years.

To find out more about Michael please refer to the following places:

Linked In: http://www.linkedin.com/in/michaelstephensonuk1

Twitter: https://twitter.com/michael_stephen

Microsoft MVP Profile: http://mvp.microsoft.com/en-us/mvp/Michael%20Stephenson-4021792

Pluralsight Author Profile: http://pluralsight.com/training/Authors/Details/michael-stephenson

Reviewers

Thanks to the following friends who reviewed this paper.

Brian Milburn

Brian Milburn has specialized in enterprise integration, since discovering BizTalk Server 2002. He has worked as an intendant consultant for over 15 years working on the Microsoft stack architecting and building enterprise solutions for the automotive, pharmaceutical and insurance industries. Currently involved in integration projects for local government, using BizTalk Server, Azure BizTalk Services and Azure Service Bus.

To find out more about Brian please refer to the below links:

Linked In: http://www.linkedin.com/profile/view?id=132291316

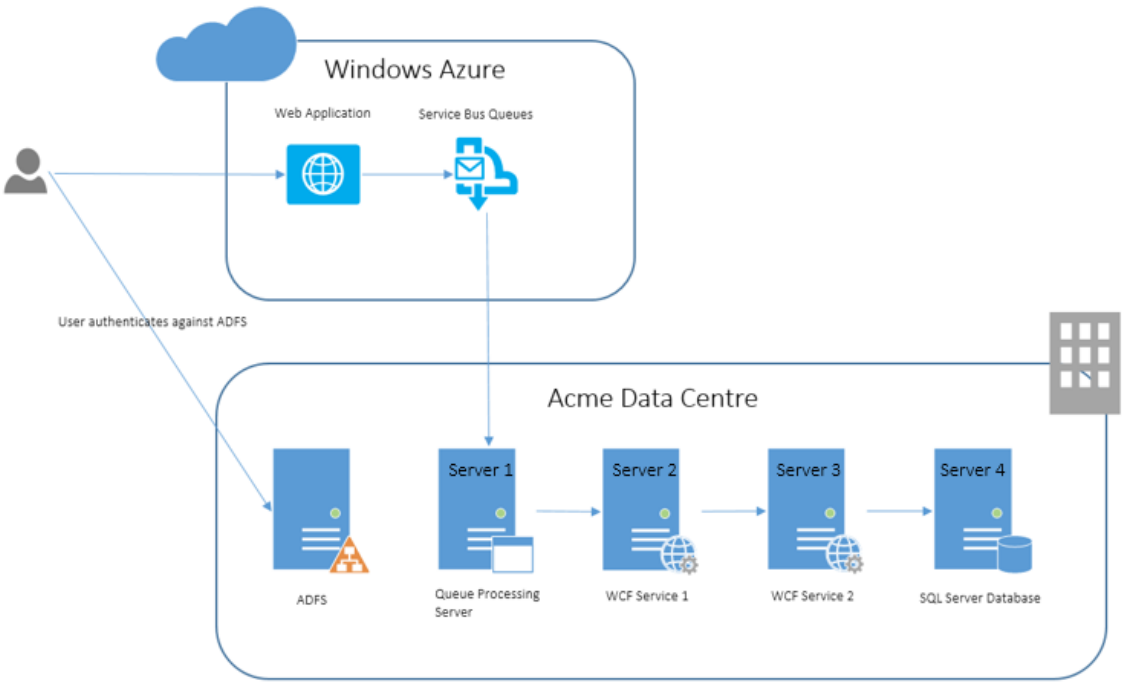
Introduction

A few years ago Paolo Salvatori wrote an excellent article about how to use Protocol Translation with BizTalk to be able to flow a windows identity through the message box in BizTalk and to flow that identity on to another service which would be able to recreate the identity and impersonate it in the back end service. I've always been a big fan of this article as it is a difficult topic and as someone who has had to implement some of these multi hop impersonation and delegation scenarios a few times I have a good appreciation of how complex it can be.

In this article I am going to assume you have some familiarization with Kerberos and WCF with Windows based security. If you are not there are many online resources on these topics.

These days in addition to working with BizTalk, I also do a lot of API and Windows Azure Service Bus work. One of the solutions I have been doing some R&D for recently involves flowing a windows identity in a hybrid integration solution using Windows Azure Service Bus Queues. Conceptually it's similar to Paolo's article but in this case I am not using BizTalk and the cloud is involved so I wanted to talk through our challenge and do a walk-through of how we achieved this.

First let's take a look at our solution.



In this solution the user would access a website which is hosted in Windows Azure but would be authenticated against an on premise instance of ADFS validating the user is part of the domain and the web application may use some claims to determine what access to the application the user is allowed. When the web application needs to access resources on premise it will send a message to the Azure Service Bus and in this case we will be using queues and topics. On premise there is a server which we

will call Server 1 which will host a Windows Service which will poll the Azure Service Bus queues and then with the messages it receives it will perform some basic mapping and forward the message onto the WCF Service 1 component which is hosted on Server 2. That service will then do a simple mapping and forward a message to WCF Service 2 which is hosted on Server 3. In that component it will implement some data access logic to access the SQL Server hosted on Server 4 to retrieve some data and a response message will be sent right back through the stack and via the specified response queue where the Web Application will be polling for a response.

At this point your probably wondering why we have so many hops, but remember this is a fictitious scenario where we want to look at how we flow identity across the stack rather than what the main functionality is. In this case hopping across 4 servers is a pretty complex example which should cover anyone's needs! I have come across situations in the past where I have been looking for help online when implementing identify flow scenarios and found that the overly simplified examples are sometimes difficult to matchup to you real world requirements. In this walk through we have a more complex scenario which may represent something similar to the real world but the key thing is it includes delegating from one WCF service to another and also impersonating when accessing a database. These are the two most common scenarios I think you are likely to come across and if we can cover both of them in this paper then I hope you will just be able to miss out the hops you don't need rather than having to go searching online to find other resources to work out how they mash together to match your requirements.

In this example imagine that we have implemented the components as described above in the diagram and it all works fine but we have not yet implemented any identity flow code. The next step is that we want to secure the data in the SQL Database so that it is not accessed under the context of a service account. Instead we want to access the data using the context of the web site user who logged in against ADFS. At present without any impersonation the data would be retrieved under the context of the service account running WCF Service 2. Imagine we remove all of the permissions for the service accounts to access the database and only my individual user account is allowed to access the database. Going back to the start of the process in the web application I am already authenticated against ADFS so the web application is aware of my claims from ADFS and its now a question of how do we flow some information about my identity and then how do we use that to ensure that when I access the database from WCF Service 2 its accessing it as my user and not as the service account.

Before we get into the walk through lets recap on what components we have here:

Server	Hosts Component	Identity Set on	Running as Service Account	Its role is
Server-01	Queue Processing Windows Service	Windows Service	Acme\Svc_Server1	To pull messages from the Azure Service Bus and to forward them to WCF Service 1
Server-02	WCF Service 1	IIS App Pool	Acme\Svc_Server2	To receive a message from the queue processor and to map and forward the message to WCF Service 2

Server-03	WCF Service 2	IIS App Pool	Acme\Svc_Server3	To receive a message from WCF Service 1 and to implement some data access code to retrieve data from the database.
Server-04	SQL Server	SQL Service through SQL Install or SQL Configuration Manager	Acme\svc_sql	This server will host the SQL Server database

How will the Identity Flow work?

In the web application when the user is authenticated we will have the claims from ADFS and one of those claims (most likely the nameidentifier) will tell us what the users username from Active Directory is. For when I access the website I will have a claim which can indicate that my user is acme\stephensonm. Note that depending on how you configure ADFS you may have retrieved the claim in various formats such as the pre-windows 2000 format of acme\stephensonm or the more modern format of Michael.stephenson@acme.com. Either of these is usually fine as you will see later when we look to impersonate.

In the web application, when we send a message to the Windows Azure Service Bus we will include a context property on the message called "Identity-ActAs" which we will populate with the username value we got from ADFS. When the message is received from the Azure Service Bus queue the queue processing component will implement some C# code to use the S4U extensions to Kerberos to login as that user and to impersonate it before calling WCF Service 1. If we add some additional configuration to our Active Directory setup this identity which is sent to WCF Service 2 will support delegation and can be flown across each hop so that when we access the database using Windows Integrated Security the code will be running as me and I will be able to retrieve data rather than get an error which the service account would get.

Now let's take a look at the walk through of how we would change things to support the identity flow.

Setting up the Domain

To begin with we need to setup some stuff in Active Directory for this solution to work.

Setting Up Service Accounts in Active Directory

In the Active Directory Users and Computers module for the domain we need to ensure we have service accounts for our components to run as. Often when I have seen people try to implement these delegation scenarios they try to use the same service account for multiple components/hops and I would recommend that is something you do not do. The reason for that is it is difficult to work out exactly what settings should be where if you get something wrong. Using a separate account per component is a good

practice anyway but especially in this scenario where you can be sure of the configuration for each hop. The below table shows the service accounts I setup and what they were used for.

Username	Used for
acme\svc_server1	This account is used as the identity to run the service bus listening windows service which will run on server 1
acme \svc_server2	This account will be used as the identity of the app pool running the WCF service on server 2
acme \svc_server3	This account will be used as the identity of the app pool running the WCF service on server 3
Acme\svc_SQL	This account will be used as the identity running the SQL Server process on server 4

Setting up SPN's

The next step is to create an SPN for each service. This is an important step for the Kerberos magic to work. It is also an area people often get wrong. The list below shows what needs to be setup. You can use the SETSPN command to do this.

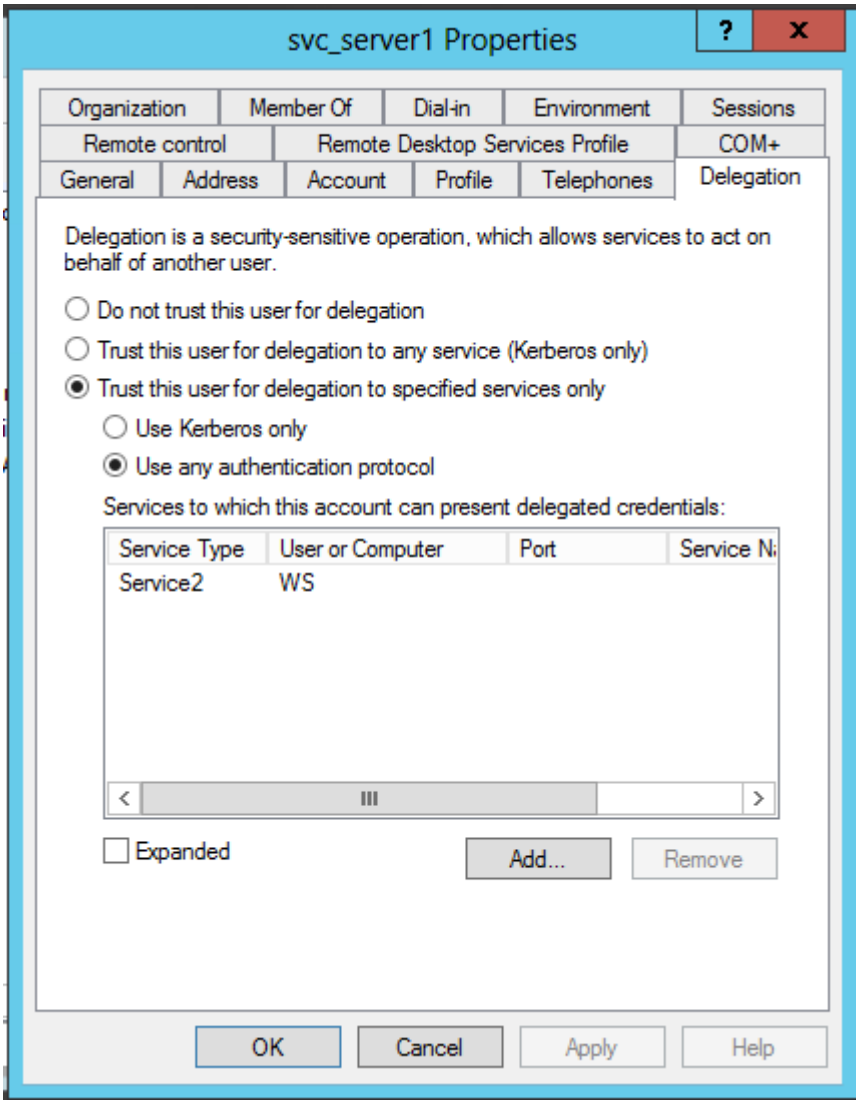
SPN	Against Object
Service1/WS	acme\svc_server1
Service2/WS	acme \svc_server2
Service3/WS	acme \svc_server3
MSSQLSvc/Server-04.acme.com:1433	Acme\svc_SQL

A couple of points to note here are that for the custom components you should create a unique SPN name that is representative for the logical component. I often use the name of the service then '/' and WS to indicate it's a web service component. This is just something I've found helpful in the past. Sometimes people call them things like the name of the machine and other things. There are cases like the SQL Server SPN where this needs to match the machine name but in this case for the custom WCF service it doesn't matter what the SPN value is, it's just an arbitrary string. As I mentioned the SQL SPN is machine specific and if you're using a SQL cluster there is guidance online about that setup. To keep this article simpler we will keep to a single SQL Server and its SPN setup as above.

Another point to note is that when setting up an SPN you can set it against a computer OR user object in active directory. For the case where we want to implement an identity flow pattern like this we need to configure them against the user account which the service will be running as.

Configuring Delegation

Next we will need to configure the delegation privileges for each service account. You will need to open up the service account users in Active Directory and modify the delegation tab as shown in the below diagram. The table below the diagram shows the specific settings for each service account.

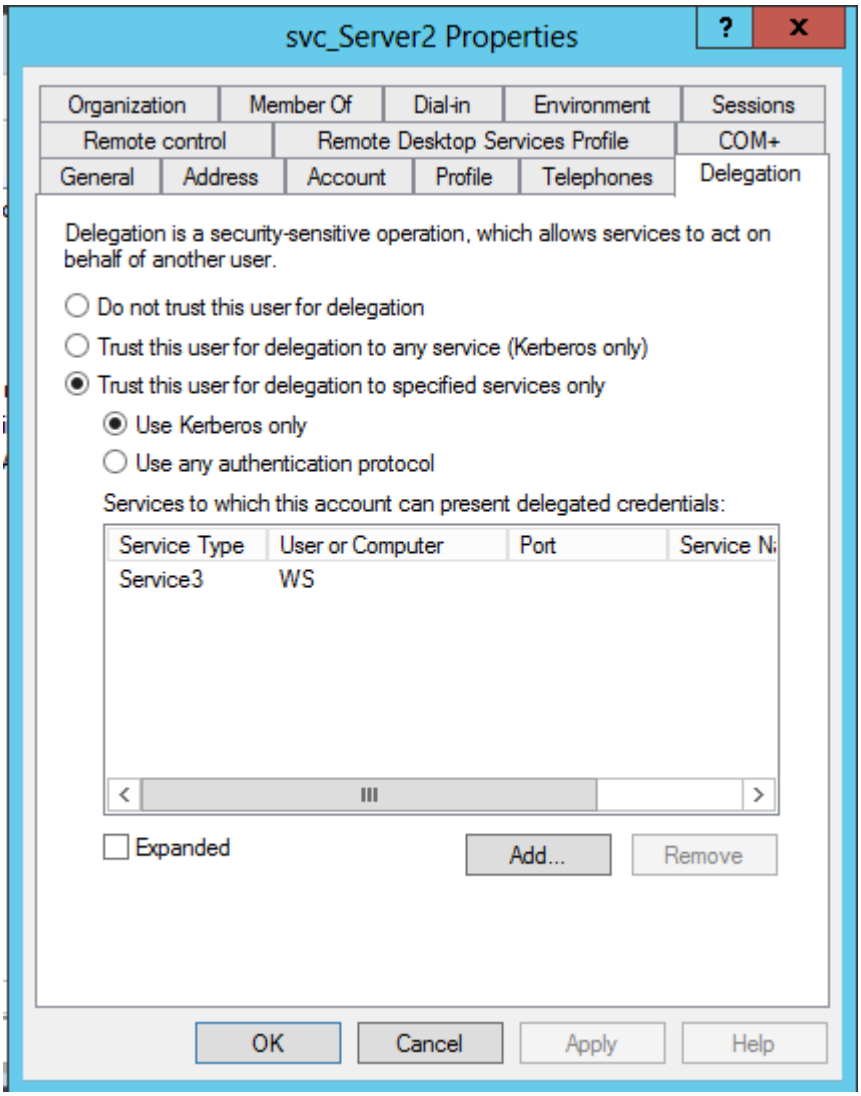


Service Account	Delegation Type	Delegation Protocol	SPNs to Delegate to
acme\svc_server1	Trust for Delegation to specific services only	Use any authentication protocol	Service2/WS
acme \svc_server2	Trust for Delegation to specific services only	Use Kerberos Only	Service3/WS
acme \svc_server3	Trust for Delegation to specific services only	Use Kerberos Only	MSSQLSvc/server-04.acme.com:1433
Acme\svc_SQL	Do not trust this user for delegation		

In addition to the picture for the Svc_Server1 account above, to check the settings for each user see the following pictures:

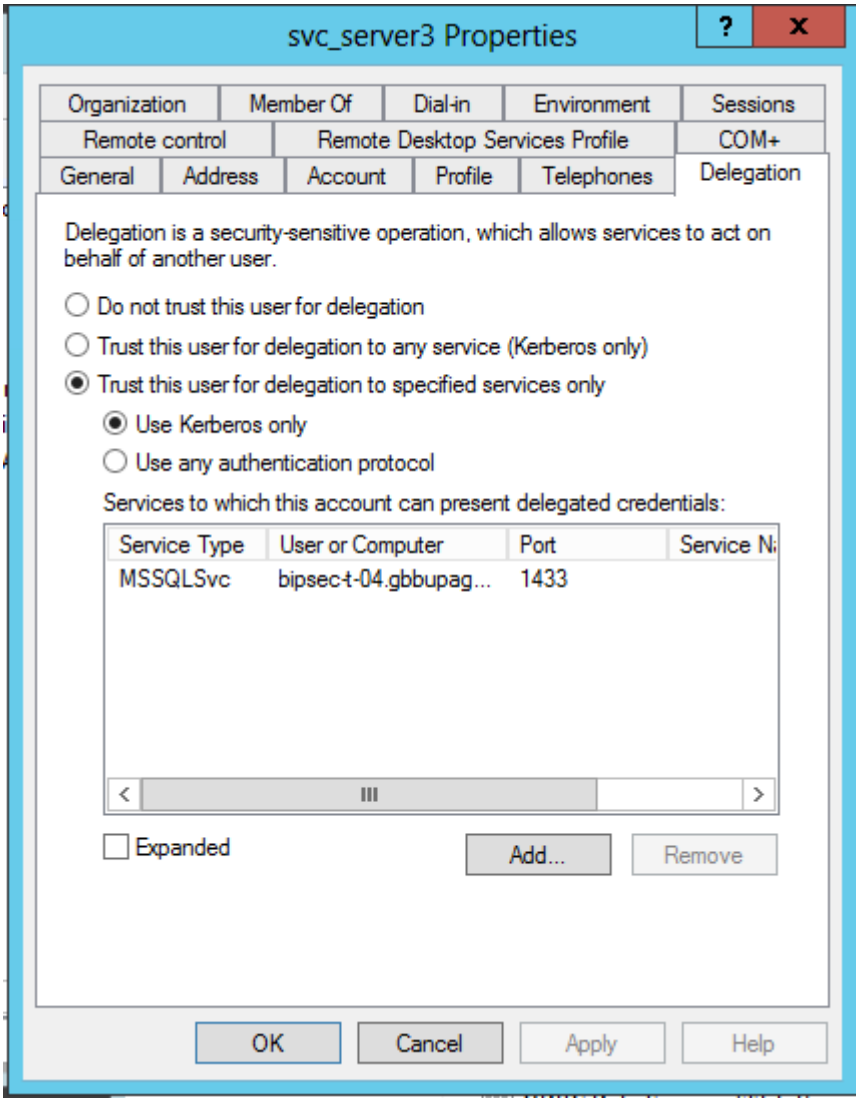
Svc_Server2

The below picture shows the delegation configuration for the svc_server2 user.



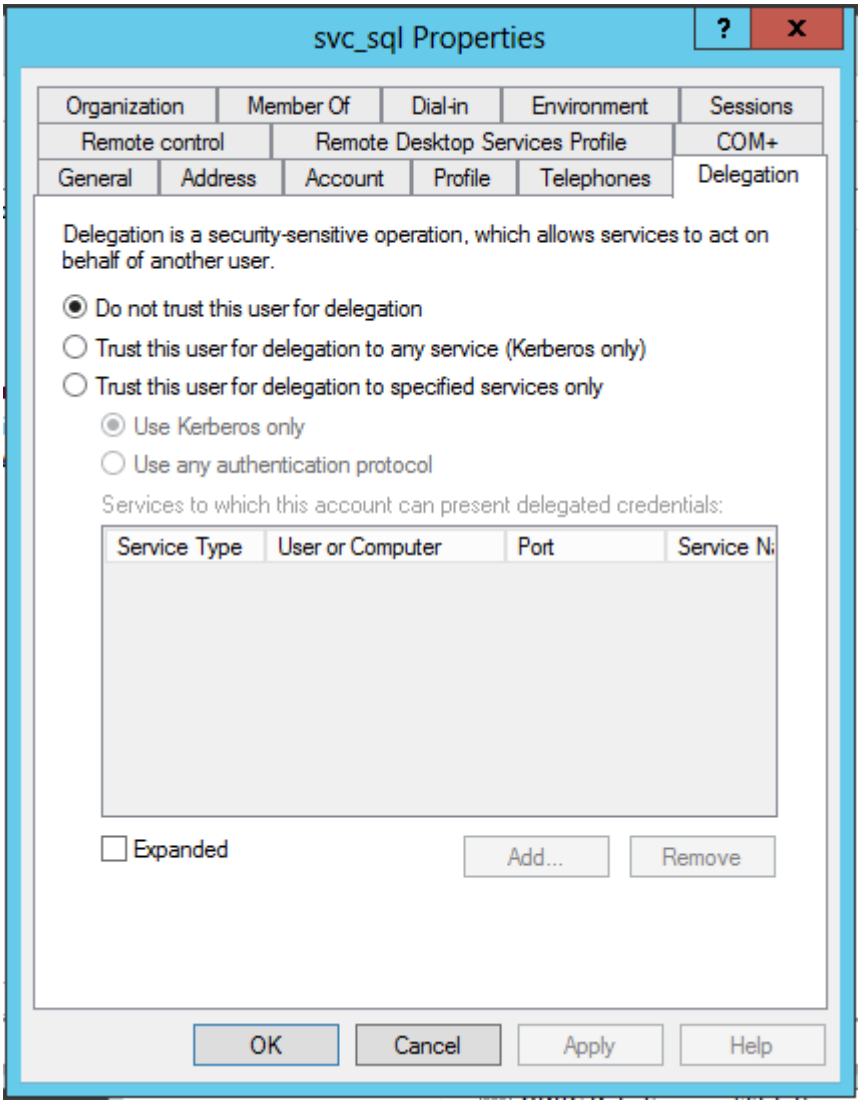
Svc_Server3

The below picture shows the delegation configuration for the svc_server3 user.



Svc_SQL

The below picture shows the delegation configuration for the svc_sql user.



Changes to Individual Servers

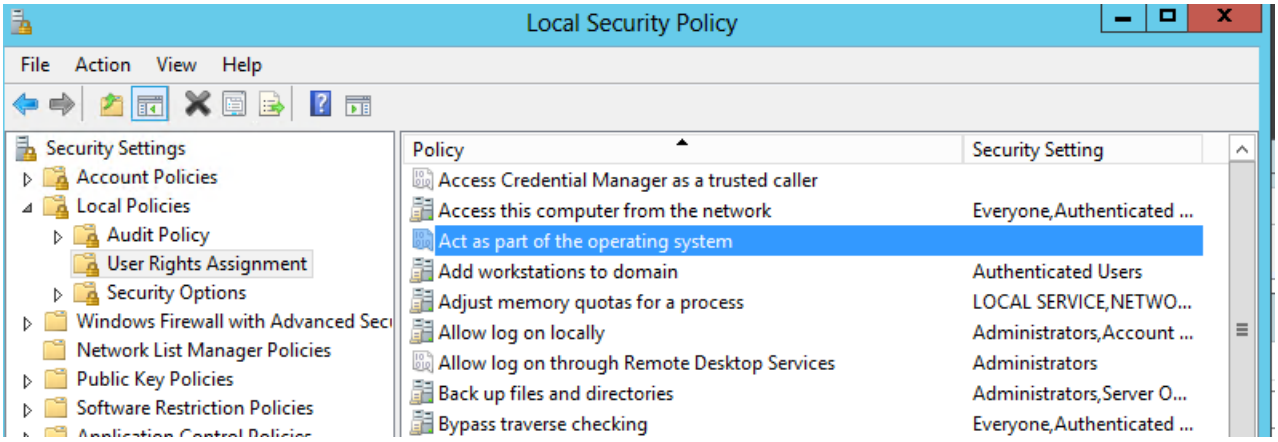
Now that we have Active Directory setup as required we will need to make a few changes to the 4 on premise servers.

Act as Part of Operating System Permission

To be able to impersonate/delegate identity the service account running a process needs to have the act as part of operating system permission. On each of the on premise servers we will need to add the act as part of operating system privilege to the service account that will be running the process on that server. To add this permission you need to take the following steps:

1. Open the Administration Tools MMC from Windows
2. Open the Local Security Policy module
3. Open the Local Policies node
4. Open the User Rights Assignment

5. Find the “Act as part of the operating system” privilege as shown below



When you open that permission you need to add the appropriate service account for each server. The below table shows which account should have this permission on each server.

Server	Service Account
Server-01	acme\svc_server1
Server-02	acme \svc_server2
Server-03	acme \svc_server3
Server-04	Acme\svc_SQL

Now that we have our infrastructure pieces in place let’s look at the code/components on each server.

Components

In this section we will discuss the key bits of each component which will make the identity flow work. I will try not to get into too much detail about the functional code which is unrelated to identity flow.

Server-01 – Queue Processing Windows Service

In this section we will look at changes we would make to our windows service which was polling the Azure Service Bus Queue. In this particular case the original code would be like any sample windows service which would be pulling messages from a queue and processing them to call a WCF service and returning a response message to the response queue, this is a standard RPC pattern using queued messaging on Windows Azure Service Bus. If you are not familiar with this pattern please refer to this article for more info which contains links to some samples which will show you how to use the AppFx.ServiceBus messaging framework to simplify this pattern for you.

Code to access Username Property

When we retrieve the message from the queue we can check the service bus message properties to see if it contains the property telling us who the code should act as. You can see a code snippet for this below.

```
	var actAs = string.Empty;
	if
(MessageProcessorContext.Current.Message.Properties.ContainsKey("Identity-
ActAs"))
	actAs = MessageProcessorContext.Current.Message.Properties["Identity-
ActAs"] as string;
```

Once we have the act as value we can use this later to impersonate.

C# Code to Impersonate

In the Windows Service Code I added some code around the call to the WCF service to do some logic to work out if we should impersonate and to do the impersonation if we should. You can see this code below.

```
            if (ImpersonationManager.ShouldImpersonate() &&
string.IsNullOrEmpty(actAs) == false)
            {
                using (var impersonationMgr = new ImpersonationManager(actAs))
                {
                    LogIdentity();
                    var client = new WCFClient.MyServiceClient("Server2-
WCFService1-MyService");
                    client.ClientCredentials.Windows.AllowedImpersonationLevel =
System.Security.Principal.TokenImpersonationLevel.Delegation;
                    client.ClientCredentials.Windows.AllowNtlm = true;
                    return client.Submit(inputMessage);
                }
            }
            else
            {
                    var client = new
WCFClient.MyServiceClient(config.EndpointConfigurationName);
                    return client.Submit(inputMessage);
            }
```

In the check if the code should impersonate its using the ImpersonationManager class to check the configuration file to see if impersonation is turned on, this means that if the component is configured to impersonate and the message from the queue has given us an “act as” value then we will attempt to impersonate that user. We will then use the normal WCF type code to specify the windows Delegation impersonation level and send the message.

Impersonation Manager

Below you will find the code for the ImpersonationManager class. This class is a pretty standard implementation of the code you would write to encapsulate the use of the WindowsIdentity.Impersonate approach to impersonating. As you can see in the above code snippet we use the ImpersonationManager in a using statement so we can be sure that our ImpersonationContext has been cleaned up before we exit the code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Security.Principal;
using System.Threading;
using System.Configuration;

namespace Acme.QueueBridge.Utilities.Security
{
    public class ImpersonationManager : IDisposable
    {
        private WindowsImpersonationContext _context;

        public ImpersonationManager(string actAs)
        {
            if (ShouldImpersonate())
            {
                var upn = GetUpn(actAs);
                var id = new WindowsIdentity(upn);
                if (id == null) return;
                _context = id.Impersonate();
            }
        }

        public string GetUpn(string actAs)
        {
            if (actAs.Contains("@"))
                return actAs;

            if (actAs.Contains("\\"))
            {
                var sections = actAs.Split('\\');
                var domainName = sections[0];
                var user = sections[1];

                return string.Format("{0}@{1}", user, domainName);
            }
            throw new ArgumentException("The act as is not in the correct
format");
        }

        public bool IsImpersonating
        {
            get { return _context != null; }
        }
```

```
        public void Dispose()
        {
            if (IsImpersonating)
            {
                _context.Undo();
                GC.SuppressFinalize(this);
            }
        }

        public static bool ShouldImpersonate()
        {
            const string Key = "Impersonate";
            bool result;
            if (bool.TryParse(ConfigurationManager.AppSettings[Key], out result))
                return result;
            else
                return false;
        }
    }
}
```

WCF Configuration for calling WCF Service 1

The below configuration will show you the WCF configuration in the queue processors configuration file to allow it to talk to WCF Service 1 using Message based security and Kerberos.

```
<system.serviceModel>
    <bindings>
      <wsHttpBinding>
        <binding name="MyBinding">
          <security mode="Message">
            <message clientCredentialType="Windows"
                     negotiateServiceCredential="true"
                     algorithmSuite="Default"
                     establishSecurityContext="false"/>
          </security>
        </binding>
      </wsHttpBinding>
    </bindings>
    <client>
      <endpoint
       name="Server2-WCFService1-MyService"
       address="http://server-02/Acme.WcfService1/MyService.svc"
       binding="wsHttpBinding" bindingConfiguration="MyBinding"
       contract="Acme.WcfService1.IMyService">
        <identity>
          <servicePrincipalName value="Service2/WS" />
        </identity>
      </endpoint>
    </client>
```

```
    <behaviors>

    </behaviors>
  </system.serviceModel>
```

That's everything done for the Queue Processing component.

Server-02 – WCF Service 1

In this section we will look at the changes that need to be made to allow WCF Service 1 to accept a message which includes the identity flown from the Queue Processor and to then flow that identity to WCF Service 2 when it calls the WCF service it hosts.

WCF Configuration to Expose the Service for Queue Processor

The below configuration shows the WCF service configuration to expose the service which will be called by the queue processor. In the configuration you can see that the SPN is specified and we are using Message based Security with a Windows Identity.

```
<services>
      <service name="Acme.WcfService1.MyService"
            behaviorConfiguration="DefaultBehaviour">
            <endpoint binding="wsHttpBinding"
                      bindingConfiguration="MyBinding"
                      contract="Acme.WcfService1.IMyService">
          <identity>
            <servicePrincipalName value="Service2/WS"/>
          </identity>
        </endpoint>
      </service>
    </services>
    <bindings>
      <wsHttpBinding>
        <binding name="MyBinding">
          <security mode="Message">
            <message clientCredentialType="Windows"
                     negotiateServiceCredential="true"
                     algorithmSuite="Default"
                     establishSecurityContext="false"/>
          </security>
        </binding>
      </wsHttpBinding>
```

WCF Client Configuration to call WCF Service 2

In the below configuration snippet you can see the configuration used as the WCF client to call WCF Service 2 hosted on Server-03. This configuration uses the same binding configuration as in the code snippet above for exposing the service.

```
<client>
```

```
      <endpoint
       name="Server3-Acme.WcfService2.MyService"
       address="http://server-03/Acme.WcfService2/MyService.svc"
       binding="wsHttpBinding" bindingConfiguration="MyBinding"
       contract="Acme.WcfService1.IMyService">
        <identity>
          <servicePrincipalName value="Service3/WS" />
        </identity>
      </endpoint>
    </client>
```

Impersonation Manager

The WCF component will also use an impersonation manager class which is similar to the one in the Queue Processor except that there is a key difference. In the queue processor case a string with an act as identity was passed into the impersonation manager whereas this time you will see in the constructor that the Windows Principal is extracted from the thread and impersonation is done using the associated Windows Identity. This thread principal is setup by WCF when the service is called and if our stack is configured correctly will be using the appropriate identity we created in the queue processor.

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Linq;
using System.Security.Principal;
using System.Text;
using System.Threading;

namespace Acme.WCFService1.Utilities
{
    public class ImpersonationManager : IDisposable
    {
        private WindowsImpersonationContext _context;

        public ImpersonationManager()
        {
            if (ShouldImpersonate())
            {
                var principal = Thread.CurrentPrincipal as WindowsPrincipal;
                if (principal == null) return;
                var id = principal.Identity as WindowsIdentity;
                if (id == null) return;
                _context = id.Impersonate();
            }
        }

        public bool IsImpersonating
        {
            get { return _context != null; }
        }

        public void Dispose()
```

```
        {
            if (IsImpersonating)
            {
                _context.Undo();
                GC.SuppressFinalize(this);
            }
        }

        private bool ShouldImpersonate()
        {
            const string Key = "Impersonate";
            bool result;
            if (bool.TryParse(ConfigurationManager.AppSettings[Key], out result))
                return result;
            else
                return false;
        }
    }
}
```

Code to call WCF Service 2

In the code for the WCF service 1 components Submit method we will map the inbound request message to a request for WCFService2 and then use the below code snippet to send the request. You can see that the WCF code is like a normal implementation which would call a WCF service with message based windows security. I have omitted the mapping of request and response messages to keep the snippet simple and to allow us to focus on the security side of things.

```
            //Request message created here

            using (var impersonation = new ImpersonationManager())
            {
                var client = new WcfClient.MyServiceClient("Server3-
Acme.WcfService2.MyService");
                client.ClientCredentials.Windows.AllowedImpersonationLevel =
TokenImpersonationLevel.Delegation;
                var response = client.Submit(request);

                //Response message mapped here

                return response;
            }
```

Server-03 – WCF Service 2

In this section we will look at the changes that need to be made to allow WCF Service 2 to accept a message which includes the identity flown from WCF Service 1 and to then use that identity when accessing the database.

WCF Configuration to Expose the Service for WCF Service 1 to call

The below configuration shows how the WCF Service is configured in WCF Service 2 to expose a service which will use a binding which requires message based security with a Windows Indentity.

```
<services>
      <service name="Server-03.Acme.WcfService2.MyService"
behaviorConfiguration="DefaultBehaviour">
        <endpoint binding="wsHttpBinding"
                  bindingConfiguration="MyBinding"
                  contract="Acme.WcfService2.IMyService">
          <identity>
            <servicePrincipalName value="Service3/WS"/>
          </identity>
        </endpoint>
      </service>
    </services>
```

The binding configuration used with this looks like the below.

```
<wsHttpBinding>
        <binding name="MyBinding">
          <security mode="Message">
            <message clientCredentialType="Windows"
                  negotiateServiceCredential="true"
                  algorithmSuite="Default"
                  establishSecurityContext="false"/>
          </security>
        </binding>
      </wsHttpBinding>
```

Impersonation Manager

WCF Service 2 has an implementation of the exact same Impersonation Manager class that has been used in WCF Service 1.

Code to access database

In the below code snippet which is the code to access the SQL Server database you can see how we have used the Impersonation Manager class in a using statement around the data access code. This will mean that we will impersonate the provided identity when accessing the database.

```
            using (var impersonation = new ImpersonationManager())
            {
                Log.Info(string.Format("Is impersonating: {0}",
impersonation.IsImpersonating));
                LogIdentity();
```

```
                using (var sqlConnection = new SqlConnection(connectionString))
                {
                    sqlConnection.Open();
                    var cmd = new SqlCommand("Select * from Customers",
sqlConnection);
                    var reader = cmd.ExecuteReader();

                    while (reader.Read())
                    {
                        var fakeMember = new FakeMember
                        {
                            BupaMemberId =
Convert.ToString(reader["CustomerID"]),
                            Title = Convert.ToString(reader["Title"]),
                            FirstName = Convert.ToString(reader["FirstName"]),
                            Surname = Convert.ToString(reader["Surname"])
                        };
                        response.Members.Add(fakeMember);
                    }
                    sqlConnection.Close();
                    reader.Dispose();
                    cmd.Dispose();
                }
            }
```

Remember that I mentioned earlier that the database was originally setup so that the service accounts do not have access to the data but my end user account has read and write access. Although I have not specifically gone through the setup of that task in this document it is quite easy to do that and I'm assuming if your reading this paper your more than comfortable with setting up SQL permissions.

Server-04 – SQL Database

On server 04 we didn't need to make any changes to the database implementation. The database already had no permissions for the service account user and had appropriate read and write permissions for my user to be able to access the data.

Web Application

In this article I wanted to focus on the identity flow from the perspective of flowing an identity through Windows Azure Service Bus and down into on premise back end applications. There are many articles on-line which will talk about connecting your web application to ADFS so I will consider this area beyond the scope of this article. That said the key thing to ensure in relation to this article is that in the web application when you send a message to the Azure Service Bus you add a message property to the message called `Identity-ActAs` which includes the value of the fully qualified username which will have been provided by ADFS in the form of a claim.

Walk Through Summary

With the above changes in place you should be able to make a request from the web application which will flow your username through Azure Service Bus as an Act-As property and then impersonate this identity and delegate the credential through multiple WCF hops and into the SQL Database.

As you can see setting this up isn't a trivial task and there are many places where this can be configured incorrectly which is one of the key reasons I wrote this paper.

Extending this Capability

In this section we will consider some other things you might be thinking about after reading this article so far.

What about BizTalk?

If we now think back to Paolo's original article about BizTalk from a few years ago and consider how BizTalk might play in this architecture if the message was received from the Azure Service Bus by BizTalk. Well in fact most of the article is still completely relevant in this case. Instead of getting the identity via the WCF call to BizTalk, BizTalk would see this Identity-ActAs string on the message properties and on the inbound pipeline you would need to set this property to be the value of the BTS.WindowsUser context property so that it would flow with your message through BizTalk. When you wanted to call an external WCF service you would be able to use the same approach Paolo used with a WCF custom channel which would make BizTalk impersonate that identity before sending the message.

You would need to make sure that your BizTalk host instances identity has the right delegate to configuration setup like we did earlier in this article when we configured which Active Directory Users can delegate to which SPN's

What about other clients?

Another thought you may have at this point is what about a client which isn't a Web Application hooked into ADFS. To implement the pattern here from a client perspective you simply need to be able to send a message to Azure Service Bus and to get the response and also provide the Identity-ActAs message property which matches the username you would like to impersonate.

While this makes it a simple way to use this approach from many clients, there is obviously a lot of risk of an elevation of privilege if you are not careful with this. The first recommendation I would make is to ensure your client application is attached to your domain or is authenticating via ADFS so you have some trust that the user has been authenticated by the client application. You should then probably put some thinking into a pattern which would allow you to validate which application has sent the message to the Azure Service Bus so that when the queue processor component extracts the message it is able to decide if it trusts the sending application before it tries to impersonate the user. Although there is some risk associated with this approach on the positive side because you have used constrained delegation and specifically configured which SPN's your services can delegate to you have put in place a level of control over what resources can be accessed. As an example if the developer of WCF Service 2 decided to try and access another network resource in addition to the SQL Server and tried to do that while impersonating

the windows identity would not have a delegate permission to be able to access that resource because the associated SPN was not configured to be delegated to.

Conclusion

It’s been fun and quite challenging to put together this walk through as this is quite a complex topic but hopefully you will find this a very useful resource if you are trying to implement this kind of identity flow solution and want to do it in as controlled a way as possible.