

SharePoint Add-in Security



Agenda

- SharePoint Add-in Security Overview
- Configuring Add-in Permissions
- Understanding App Security Principals
- Server-to-Server (S2S) Trust Configuration
- Programming with Access Tokens
- Add-in Authentication using OAuth and ACS



Security Problems with SharePoint Solutions

- Code in farm solutions considered fully-trusted
 - By default, code runs with permissions of current user
 - Developer can call `SPSecurity.RunWithElevatedPrivileges`
 - Code runs as all-powerful **SHAREPOINT\SYSTEM** account
 - Code reverts to Windows identity of host application pool
- Sandbox solution code runs as current user
 - Code always runs with permissions of current user
 - Activation code runs as site administrator
 - No ability to elevate permissions if user is visitor



Let's start with a basic question

- What is a security principal?
 - An entity that is understood by a security system
 - An entity for which you can configure authorized access to resources
- Examples of security principals
 - User with an account in Active Directory
 - User with account in another identity management system (FBA)
 - Active Directory group or an FBA role
 - Computer which has been added to an Active Directory domain
 - SharePoint add-in (as of SharePoint 2013)



Add-in Authentication in SharePoint 2016

- SharePoint 2016 supports Add-in Authentication
 - Add-ins promoted to first class security principals
 - Add-in authentication makes add-in authorization possible
 - Add-in authentication only supported for CSOM & REST API
 - Add-in authentication not supported in custom web services
- SharePoint 2016 uses 3 basic types of app authentication
 - Internal authentication
 - External authentication using Server-to-Server (S2S) Trusts
 - External authentication using OAuth



Internal Authentication

- Internal authentication is used if the following are true
 - Incoming call targets a CSOM or REST API endpoint
 - Incoming call carries claims token with established user identity
 - Incoming call targets URL of an existing app web
- Important points about using internal authentication
 - It just works – no need to program in terms of access tokens
 - It's always used with client-side calls from pages in the app web
 - It can be used from remote web pages using cross domain library
 - It does not support app-only authentication to elevate privilege

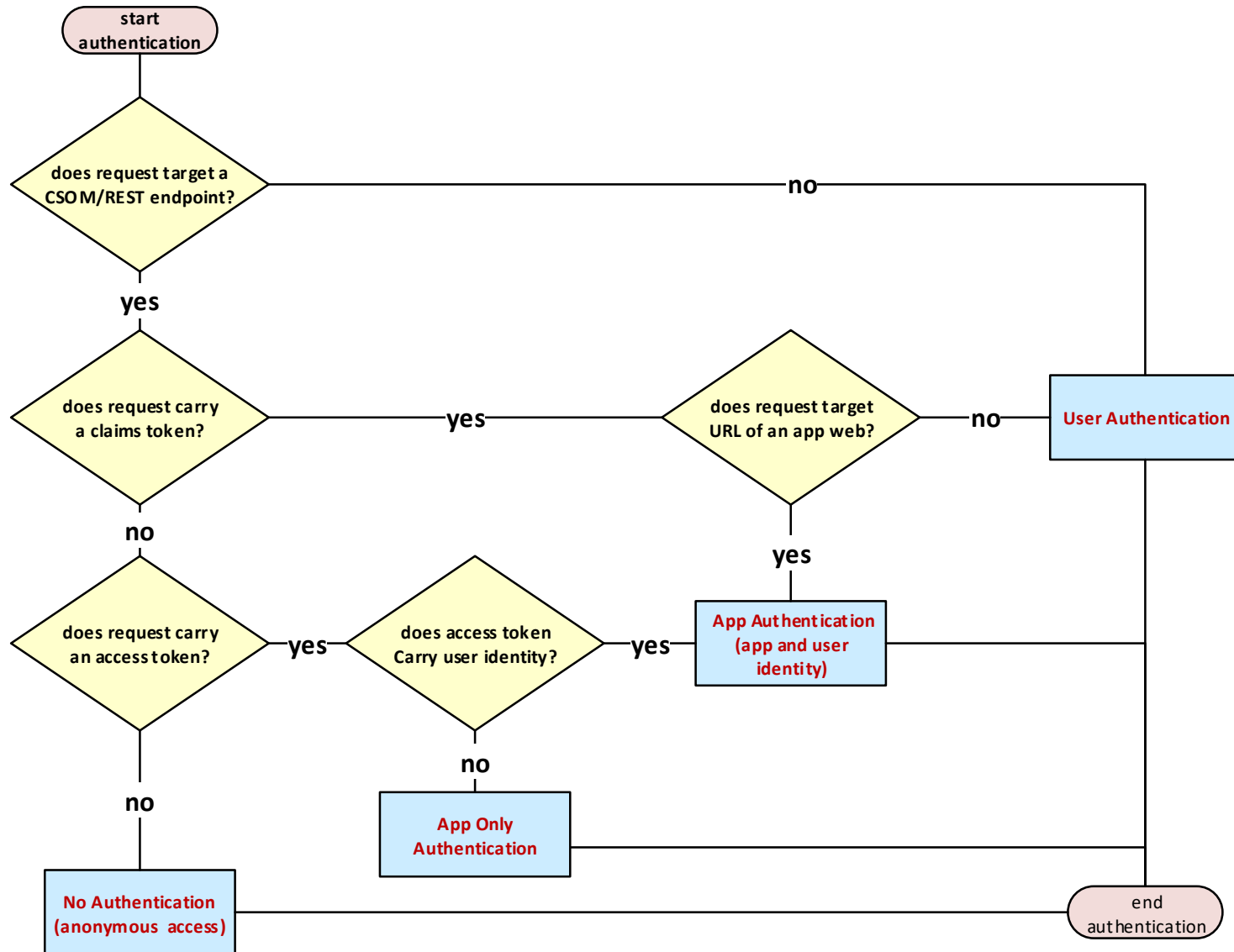


External Authentication

- In which scenarios does external authentication occur?
 - When server-side code in the remote web issues CSOM or REST API calls against the SharePoint host
 - Incoming calls free to target host web and other sites in tenancy
- How does it work?
 - App code must be written to create and manage access tokens
 - Access token carries app identity
 - Access token can (and usually does) carry user identity as well
 - App must transmit access token when calling to SharePoint



SharePoint 2016 Authentication Flow



Agenda

- ✓ SharePoint Add-in Security Overview
- Configuring Add-in Permissions
 - Understanding App Security Principals
 - Server-to-Server (S2S) Trust Configuration
 - Programming with Access Tokens
 - Add-in Authentication using OAuth and ACS



Adding Permission Requests

- Permissions requests are added to app manifest
 - App manifest designer makes this relatively easy

The screenshot shows the 'AppManifest.xml' designer window. The title is 'Permission Request Demo'. The name is 'PermissionRequestDemo'. The icon is 'PermissionRequestDemo/Images/AppIcon.png' with a 'Browse...' button. The start page is 'PermissionRequestDemo/Pages/Default.aspx'. The query string is '{StandardTokens}'. Below these fields is the 'Permission requests' section, which is expanded to show a table with columns 'Scope', 'Permission', and 'Properties'. The table contains two entries: 'Web' with 'Write' permission, and 'Site Collection' with 'Read' permission. A dropdown menu is open below the table, showing a list of scopes including 'BCS', 'Enterprise Resources', 'List', 'Micro Feed', 'Project', 'Project Server', 'Projects', 'Reporting', 'Search', 'Social Core', 'Statusing', 'Taxonomy', 'Tenant', 'User Profiles', and 'Workflow'. The 'Search' option is currently selected.

```
<AppPermissionRequests>
  <AppPermissionRequest
    Scope="http://sharepoint/content/sitecollection/web"
    Right="Write" />
  <AppPermissionRequest
    Scope="http://sharepoint/content/sitecollection"
    Right="Read" />
</AppPermissionRequests>
```

App-Only Permissions

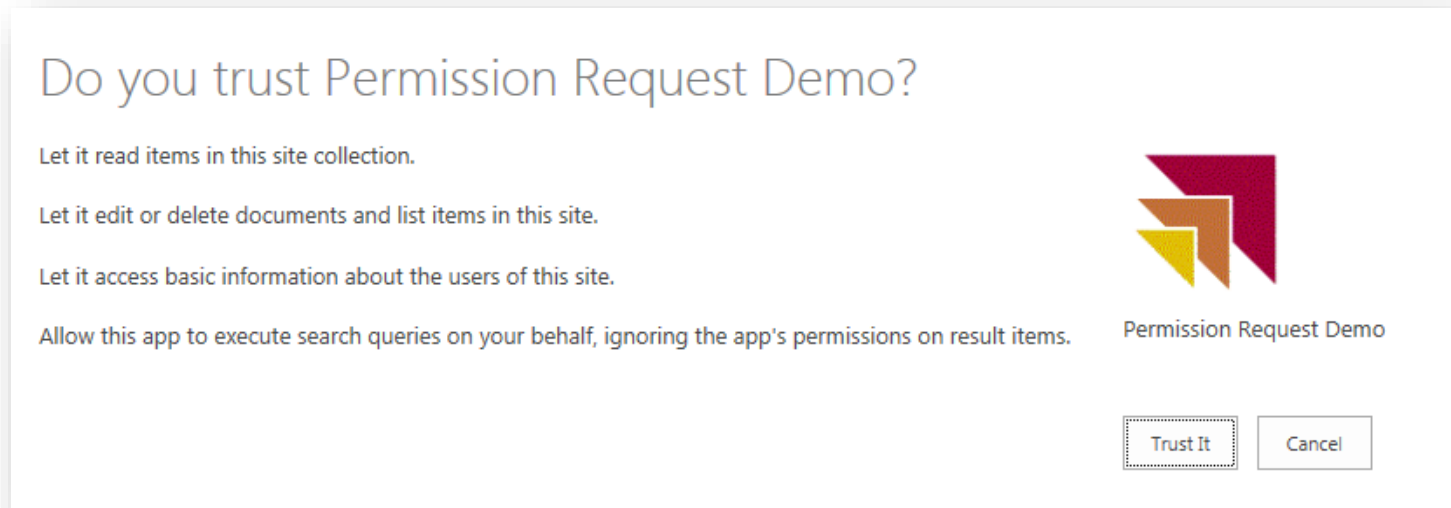
- Used for two key scenarios
 - To call into SharePoint with permissions greater than the current user (elevation)
 - To call in to SharePoint when there is no current user
- Steps to accomplish this
 - Add AllowAppOnlyPolicy to AppManifest.xml
 - Write code to acquire an app only access token

```
<AppPermissionRequests AllowAppOnlyPolicy="true">  
  <AppPermissionRequest Scope="http://sharepoint/content/sitecollection" Right="Read" />  
  <AppPermissionRequest Scope="http://sharepoint/content/sitecollection/web/list" Right="FullControl" />  
  <AppPermissionRequest Scope="http://sharepoint/search" Right="QueryAsUserIgnoreAppPrincipal" />  
</AppPermissionRequests>
```



Granting Consent in SharePoint 2016

- User prompted to trust the app during installation
 - Trust It grants requested permissions to app
 - Cancel prevents app from being installed





DEMO

Adding Permissions Request to the App Manifest

Agenda

- ✓ SharePoint Add-in Security Overview
- ✓ Configuring Add-in Permissions
- Understanding App Security Principals
 - Server-to-Server (S2S) Trust Configuration
 - Programming with Access Tokens
 - Add-in Authentication using OAuth and ACS



App Principals

- External authentication requires app principals
 - App principal is a tenancy-scoped account for app identity
 - App principal identified using a GUID
 - App principals must be created in SharePoint host
- App principal properties
 - Client ID: GUID-based identifier for app principal
 - Client Secret: (not used in S2S)
 - App Host Domain: Base URL of remote web
 - Redirect URL: URL to a page used to configure on-the-fly security



Managing App Principals in SharePoint 2016

- Get to know the built-in app management pages
 - AppRegNew.aspx
 - AppInv.com
 - AppPrincipals.aspx
- There is also management support using PowerShell
 - Use PowerShell cmdlets to administer SharePoint apps and app principals



Registering an App Security Principal

- Done automatically by Visual Studio during development
 - When you press {F5}, VS automatically registers app principal
 - Visual Studio also updates web.config file
- Can also be done using AppRegNew.aspx page
 - App deployment covered in more detail in App Publishing module

App Information
The app's information, including app id, secret, title, hosting url and redirect url.


App Id:

App Secret:


Title:

App Domain:
Example: "www.contoso.com"

Redirect URI:
Example: "https://www.contoso.com/default.aspx"



Home

 ⓘ

Recent

- Calculator App
- Apps for SharePoint
- Apps for Office
- App Requests
- Site Contents

The app identifier has been successfully created.

App Id: 22222222-2222-2222-2222-222222222222
App Secret: BbHUyA5x5vwULw+/oKB6jh7VNklkZjSRbGbTOMRaNSE=
Title: Wingtip Search App
App Domain: searchapp.wingtip.com
Redirect URI:





DEMO

Registering an App Principal

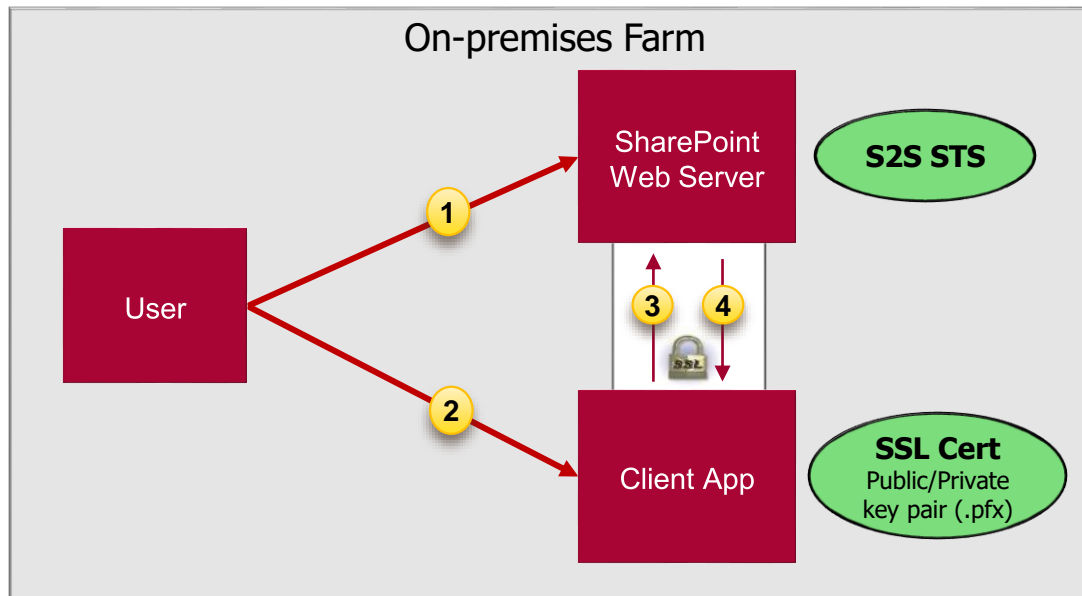
Agenda

- ✓ SharePoint Add-in Security Overview
- ✓ Configuring Add-in Permissions
- ✓ Understanding App Security Principals
- Server-to-Server (S2S) Trust Configuration
 - Programming with Access Tokens
 - Add-in Authentication using OAuth and ACS



What is a Server-to-server (S2S) Trust

- Trusted connection between Add-in and SharePoint
 - Eliminates need for ACS when running apps in on-premises farm
 - Trust between servers configured using SSL certificates
 - Requires registering public key with local SharePoint farm
 - Add-in code requires access to password and private key file



Why Is It Called a “High Trust” App

- "High trust" means that add-in must authenticate the user
 - Add-in must first authenticate user
 - Add-in must create access token and embed user identity
 - Add-in passes access token to SharePoint with each call
 - SharePoint validates access token using public key
 - SharePoint farm trusts add-in is telling the truth about user identity
- “High Trust” is very different from “Full Trust”
 - Full trust code is not limited by permissions – it can do anything
 - High trust app has set of permissions that say what it can do



Configuring a Server-to-Server Trust

- Steps to configure an S2S trust
 1. Create an x509 certificate
 2. Export public key (.cer) and password-protected private key (.pfx)
 3. Configure certificate as a trusted root authority
`New-SPTrustedRootAuthority`
 4. Use public key to create a trusted security token issuer
`New-SPTrustedSecurityTokenIssuer`
 5. Make password and private key file accessible to remote web



Creating Certificates

CreateTestCertificateForS2STrust.ps1 X

CLS

```
# create variable to call to makecert.exe command-line utility
$makecert = $PSScriptRoot + "\makecert.exe"

$certname = "WingtipAppCertificate01"
$password = ConvertTo-SecureString "Password1" -AsPlainText -Force
$startdate = (Get-Date).ToString("MM/dd/yyyy")
$enddate = ((Get-Date).AddYears(2)).ToString("MM/dd/yyyy")

# delete any pre-existing certificates with same name
Get-ChildItem Cert:\CurrentUser\My | ? {$_.Subject -eq "CN=$certname"} | Remove-Item

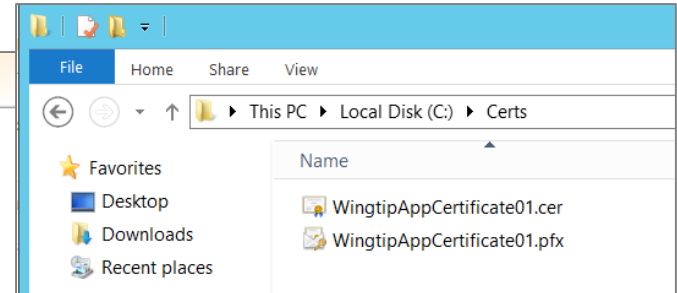
Write-Host "Creating new x509 Certificate with subject name of $certname"
$silentResult = & $MakeCert -r -pe -n "CN=$certname" -b $startdate -e $enddate -ss my -eku 1.3.6.1.5.5.7.3.1

$cert = Get-ChildItem Cert:\CurrentUser\My | ? {$_.Subject -eq "CN=$certname"}

# create local directory to export SSL certificate files
$outputDirectory = "c:\Certs\"
New-Item $outputDirectory -ItemType Directory -Force -Confirm:$false | Out-Null

$publicCertificatePath = $outputDirectory + $certname + ".cer"
Write-Host "Exporting public key to $publicCertificatePath"
$result = Export-Certificate -Type CERT -FilePath $publicCertificatePath -Cert $cert -Force

$privateCertificatePath = $outputDirectory + $certname + ".pfx"
Write-Host "Exporting password-protected private key to $privateCertificatePath "
$result = Export-PfxCertificate -FilePath $privateCertificatePath -Cert $cert -Password $password -Force
```



Creating the Secure Token Issuer

CreateTrustedSecurityTokenIssuer.ps1 X

```
Add-PSSnapin "Microsoft.SharePoint.PowerShell"

$issuerID = "11111111-1111-1111-1111-111111111111"

# remove any pre-existing with the same issuer ID
Get-SPTrustedSecurityTokenIssuer | where {$_.Name -eq $issuerID } | Remove-SPTrustedSecurityTokenIssuer -Confirm:$false

# get GUID for current SharePoint tenancy
$targetSiteUrl = "http://wingtipserver"
$targetSite = Get-SPSite $targetSiteUrl
$realm = Get-SPAAuthenticationRealm -ServiceContext $targetSite

# parse together RegisteredIssuerName value
$registeredIssuerName = $issuerID + '@' + $realm

$publicCertificatePath = "C:\Certs\WingtipAppCertificate01.cer"
$publicCertificate = Get-PfxCertificate $publicCertificatePath

Write-Host
Write-Host "Using .cer file to register certificate as root authority with local SharePoint farm"
# this is new requirement for SharePoint 2016 - not required in SharePoint 2013
$silentResult = New-SPTrustedRootAuthority -Name "WingtipAppCertificate01" -Certificate $publicCertificate

Write-Host
Write-Host "Using .cer file to register trusted token issuer in local SharePoint farm"
$secureTokenIssuer = New-SPTrustedSecurityTokenIssuer `
    -Name $issuerID `
    -RegisteredIssuerName $registeredIssuerName `
    -Certificate $publicCertificate `
    -IsTrustBroker

$secureTokenIssuer | Format-List Id, Name, RegisteredIssuerName
$secureTokenIssuer | Format-List Id, Name, RegisteredIssuerName, SigningCertificate | Out-File -FilePath "SecureTokenIssuer.txt"

# configure SharePoint to support S2S Trusts with HTTP in addition to HTTPS
$serviceConfig = Get-SPSecurityTokenServiceConfig
$serviceConfig.AllowOAuthOverHttp = $true
$serviceConfig.Update()
```

Creating an S2S App Principal

- Can be done several different ways
 - Use built-in page named `AppRegNew.aspx`
 - Use `Register-SPAppPrincipal`
 - Use `SPAppPrincipalManager`
 - Let Visual Studio do it for you when developing

```
Add-PSSnapin "Microsoft.SharePoint.PowerShell"

# set initialization values for new app principal
$appDisplayName = "My First S2S App"
$clientID = "22222222-2222-2222-2222-222222222222"

$targetSiteUrl = "http://wingtipserver"
$targetSite = Get-SPSite $targetSiteUrl
$realm = Get-SPAuthenticationRealm -ServiceContext $targetSite
$fullAppPrincipalIdentifier = $clientID + '@' + $realm

Write-Host "Register new app principal"
$registeredAppPrincipal = Register-SPAppPrincipal -NameIdentifier $fullAppPrincipalIdentifier
                                     -Site $targetSite.RootWeb
                                     -DisplayName $appDisplayName

$registeredAppPrincipal | select * | Format-List
$registeredAppPrincipal | select * | Format-List | Out-File -FilePath "SecurityPrincipal01.txt"
```





DEMO

Configuring a Secure Token Issuer

Agenda

- ✓ SharePoint Add-in Security Overview
- ✓ Configuring Add-in Permissions
- ✓ Understanding App Security Principals
- ✓ Server-to-Server (S2S) Trust Configuration
- Programming with Access Tokens
 - Add-in Authentication using OAuth and ACS



Configuring the S2S Certification in VS

- Visual Studio provides two app authentication options
 - Use Windows Azure Access Control Service (*this means OAuth*)
 - Use a certificate (*this means S2S*)

```
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.5" />
    <httpRuntime targetFramework="4.5" />
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>

  <appSettings>
    <add key="ClientId" value="22222222-2222-2222-2222-222222222222" />
    <add key="ClientSigningCertificatePath" value="C:\Certs\WingtipAppCertificate01.pfx" />
    <add key="ClientSigningCertificatePassword" value="Password1" />
    <add key="IssuerId" value="11111111-1111-1111-1111-111111111111" />
  </appSettings>

  <system.serviceModel>...</system.serviceModel>
</configuration>
```

New app for SharePoint

Configure authentication settings

How do you want your app to authenticate?

☐ Use Windows Azure Access Control Service (for SharePoint cloud apps)

☒ Use a certificate (for SharePoint on-premises apps using high-trust)

Certificate location:

C:\Certs\WingtipAppCertificate01.pfx

Password:

Issuer ID:

11111111-1111-1111-1111-111111111111

< Previous Next > Finish Cancel

App Manifest during Development

```
<App xmlns="http://schemas.microsoft.com/sharepoint/2012/app/manifest"  
  Name="MyFirstS2SAppSolution"  
  ProductID="{b5e9a36b-33c1-4dbf-b69f-d1ef8b8ed7f2}"  
  Version="1.0.0.0"  
  SharePointMinVersion="15.0.0.0" >  
  
  <Properties>  
    <Title>My First S2S App Solution</Title>  
    <StartPage>~remoteAppUrl/Pages/Default.aspx?{StandardTokens}</StartPage>  
  </Properties>  
  
  <AppPrincipal>  
    <RemoteWebApplication ClientId="*" />  
  </AppPrincipal>  
  
  <AppPermissionRequests>  
    <AppPermissionRequest Scope="http://sharepoint/content/sitecollection/web" Right="Read" />  
  </AppPermissionRequests>  
  
</App>
```



App Manifest in Real-world Deployment

```
<App xmlns="http://schemas.microsoft.com/sharepoint/2012/app/manifest"  
  Name="MyFirstS2SAppSolution"  
  ProductID="{b5e9a36b-33c1-4dbf-b69f-d1ef8b8ed7f2}"  
  Version="1.0.0.0"  
  SharePointMinVersion="15.0.0.0" >  
  
  <Properties>  
    <Title>My First S2S App Solution</Title>  
    <StartPage>https://remoteweb1.wingtip.com/Pages/Default.aspx?{StandardTokens}</StartPage>  
  </Properties>  
  
  <AppPrincipal>  
    <RemoteWebApplication ClientId="22222222-2222-2222-2222-222222222222" />  
  </AppPrincipal>  
  
  <AppPermissionRequests>  
    <AppPermissionRequest Scope="http://sharepoint/content/sitecollection/web" Right="Read" />  
  </AppPermissionRequests>  
  
</App>
```



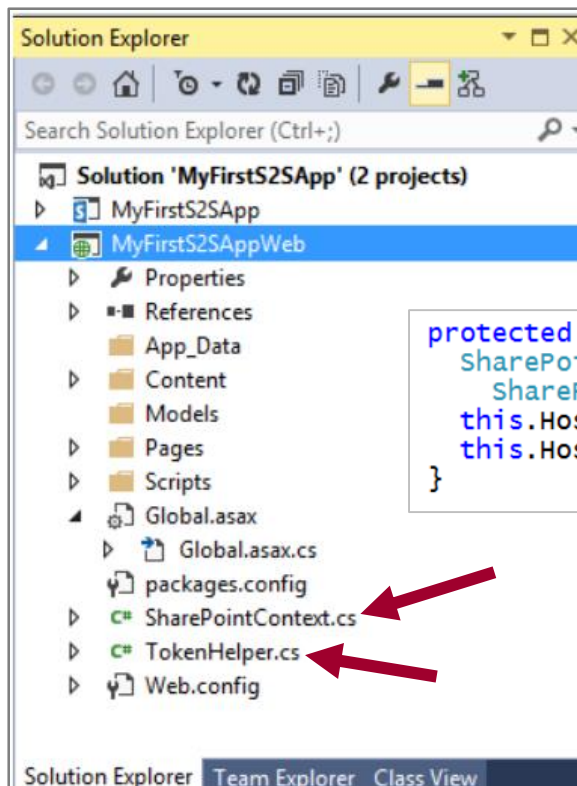
Developing Add-ins that use S2S Trusts

- What is developer responsible for with Add-in using S2S?
 - Authenticating the user (can use Windows Auth, FBA, etc.)
 - Create access tokens
 - Transmit access token with every call to CSOM or REST API
- What's an S2S access token?
 - S2S access token created using format in OAuth2 specification
 - S2S access token contains app identity (client ID)
 - S2S access token contains issuer identity (issuer ID)
 - S2S access token can optionally include user identity
 - S2S access token must be signed using PFX key of SSL certificate



Programming with Access Tokens

- Visual Studio adds two utility helpful classes
 - TokenHelper
 - SharePointContext



```
protected void Page_Load(object sender, EventArgs e) {  
    SharePointContext spContext =  
        SharePointContextProvider.Current.GetSharePointContext(Context);  
    this.HostWebLink.NavigateUrl = spContext.SPHostUrl.AbsoluteUri;  
    this.HostWebLink.Text = "Back to Host Web";  
}
```



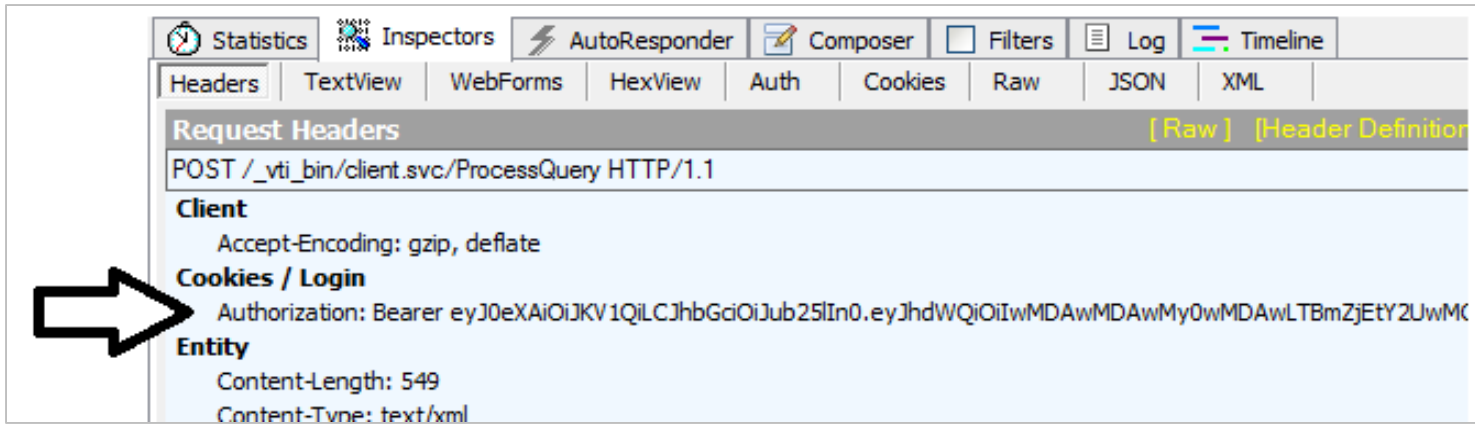
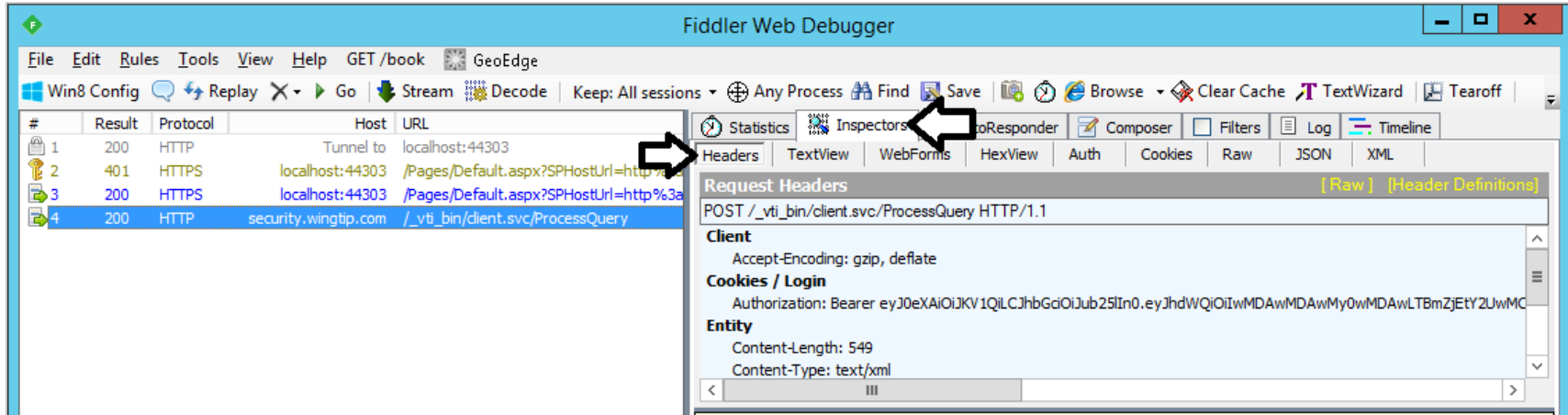
CSOM Calls using S2S Authentication

- **TokenHelper** class has methods specific to S2S
- **SharePointContext** has methods that are not S2S-specific

```
protected void cmdGetTitleCSOM_Click(object sender, EventArgs e) {  
    SharePointContext spContext =  
        SharePointContextProvider.Current.GetSharePointContext(Context);  
  
    using (var clientContext = spContext.CreateUserClientContextForSPHost()) {  
        // make CSOM call to SharePoint host  
        clientContext.Load(clientContext.Web);  
        clientContext.ExecuteQuery();  
        placeholderMainContent.Text = "Host web title (CSOM): " + clientContext.Web.Title;  
    }  
}
```




Examining S2S CSOM Calls using Fiddler



REST Calls using S2S Authentication

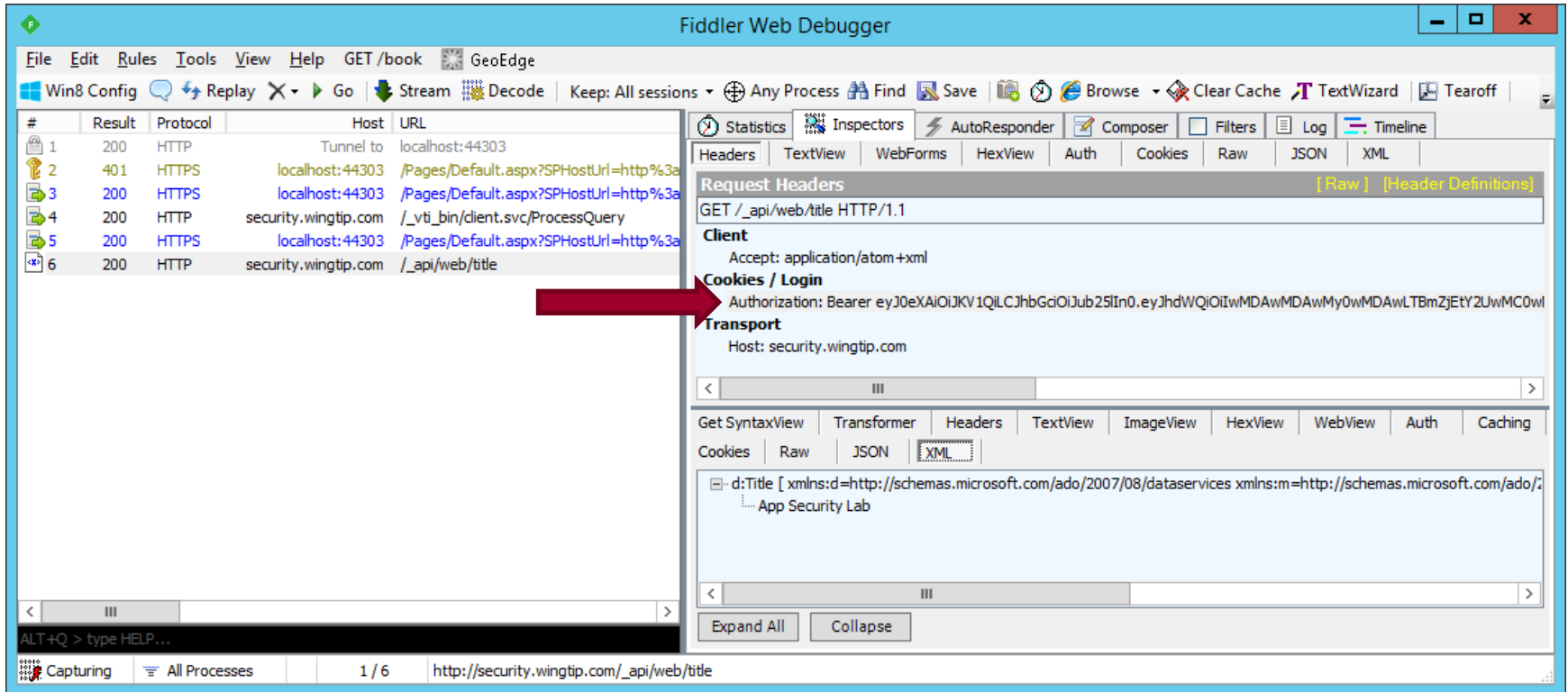
- Authorization header must be added explicitly



```
protected void cmdGetTitleREST_Click(object sender, EventArgs e) {  
    SharePointContext spContext =  
        SharePointContextProvider.Current.GetSharePointContext(Context);  
  
    string restUri = spContext.SPHostUrl + "_api/web/title";  
  
    HttpWebRequest request = WebRequest.Create(restUri) as HttpWebRequest;  
    request.Accept = "application/atom+xml";  
  
    string spAccessToken = spContext.UserAccessTokenForSPHost;  
    request.Headers["Authorization"] = "Bearer " + spAccessToken;  
  
    HttpWebResponse response = request.GetResponse() as HttpWebResponse;  
    XDocument responseBody = XDocument.Load(response.GetResponseStream());  
  
    XNamespace nsDataService = "http://schemas.microsoft.com/ado/2007/08/dataservices";  
    string hostWebTitle = responseBody.Descendants(nsDataService + "Title").First().Value;  
  
    placeholderMainContent.Text = "Host web title (REST): " + hostWebTitle;  
}
```



Examining S2S REST Calls using Fiddler



The background of the slide is a close-up, low-angle shot of a server rack. The server units are illuminated with bright blue light, creating a strong sense of depth and perspective. The lights are arranged in vertical columns, and the overall color palette is dominated by deep blues and bright cyan highlights.

DEMO

Creating a Provider-hosted App which uses S2S Authentication

Agenda

- ✓ SharePoint Add-in Security Overview
- ✓ Configuring Add-in Permissions
- ✓ Understanding App Security Principals
- ✓ Server-to-Server (S2S) Trust Configuration
- ✓ Programming with Access Tokens
- Add-in Authentication using OAuth and ACS



External Authentication using ACS

- External authentication can use OAuth and ACS
 - ACS = Azure Access Control Services
 - Requires configuring trust in local farm to ACS servers

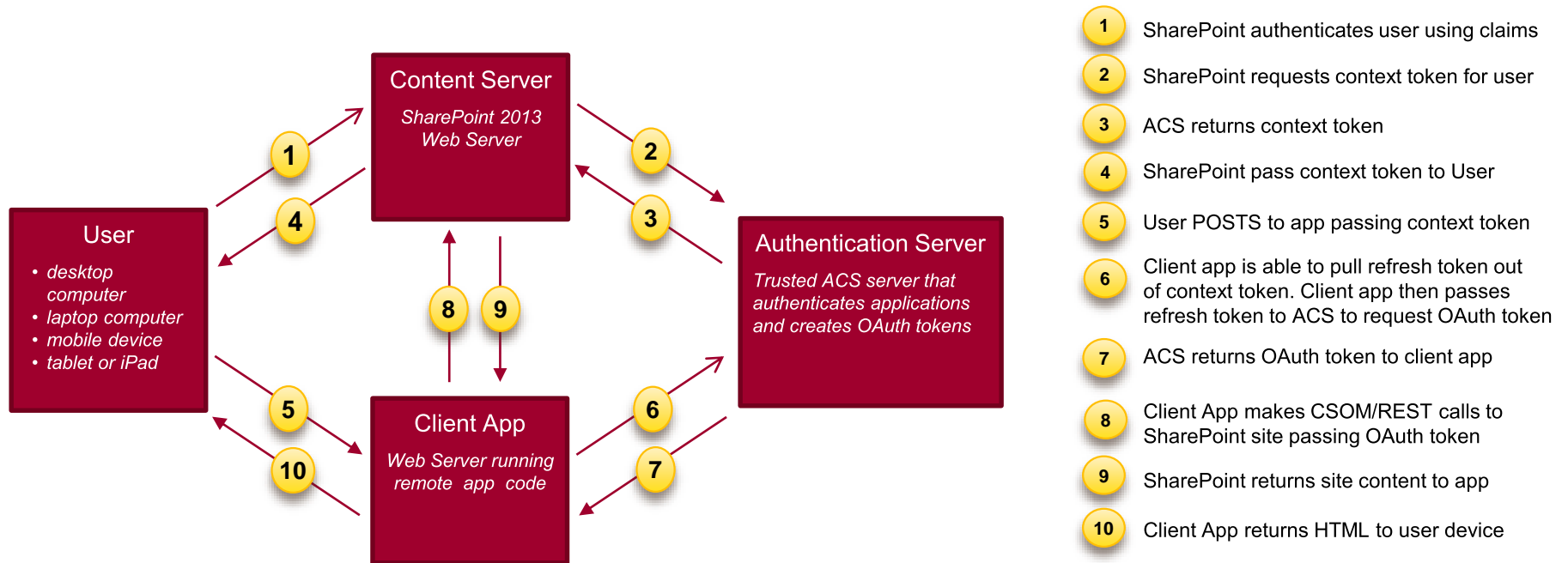


Security Tokens used in OAuth

- Context Token
 - Contextual information passed to app
- Refresh Token
 - Used by client app to acquire an access token
- Access Token
 - Token passed to SharePoint to app when using external authentication
- Authorization Code
 - Used to register an app with on the fly permissions



OAuth Protocol Flow in SharePoint 2016



Summary

- ✓ SharePoint Add-in Security Overview
- ✓ Configuring Add-in Permissions
- ✓ Understanding App Security Principals
- ✓ Server-to-Server (S2S) Trust Configuration
- ✓ Programming with Access Tokens
- ✓ Add-in Authentication using OAuth and ACS

