Siva's Blog

Monday, February 24, 2014

PowerShell Automation to Give AWS Console Access

If your organization supports SAML, you can let users who have been authenticated in your organization, access the AWS Management Console without having to have IAM identities and without having to sign in again. AWS provides a single sign-on (SSO) endpoint (https://signin.aws.amazon.com/saml) that accepts SAML assertions that are used to grant your users federated access to the console. You can find info here.

In this blog, I will focus on PowerShell automation to setup Active Directory, Active Directory Federation Service, configure the AWS roles etc. to achieve end to end automation. Manual configuration of this is nicely explained here.

Overview

1. The user browses to your organization's portal and selects the option to go to the AWS Management Console.
2. The portal verifies the user's identity in your organization.
3. The portal generates a SAML authentication response that includes assertions that identify the user and include attributes about the user.
4. The client posts the SAML assertion to an AWS single sign-on endpoint. The endpoint uses the AWS STS AssumeRoleWithSAML API to request temporary security credentials and creates a console sign-in URL.
5. AWS sends the sign-in URL back to the client with a redirect.
6. The client gets the console sign-in and is redirected to the AWS Management Console.

Prerequisites

- Sign up for AWS and get the AccessKey & SecretKey. You can find the info about AWS Account and Access Keys here.
- Install PowerShell module from here. All the AWS tools for Windows are nicely bundled as a single MSI. Setup instructions can be found here. Call Initialize-AWSDefaults to setup the default AccessKey and SecretKey to use.

- Initialize the script variables, these variables are used throughout the script.

```
#location where makecert tool is present, you can find this tool part of Windows SDK
$makecertpath = 'c:\temp\makecert.exe'
$domainname = "sivadomain.com"
$netbiosname = "sivadomain"
#for illustration password is hardcoded, best is to use Get-Credential
$password = ConvertTo-SecureString "Password123" -AsPlainText -Force
```

- Code to do prerequisite tests

```
if ( !(Test-Path -Path $makecertpath))
{
  'makecertpath should point to full path of makecert.exe'
  return
}

if ((Get-Module AWSPowerShell) -eq $null)
{
  'AWSPowerShell is not installed'
  return
}

if ((Get-AWSCredentials -ListStoredCredentials | Select-String 'AWS PS Default') -eq $null)
{
  'AWS PS Default is not set. Run Initialize-AWSDefaults'
  return
}
```

Install Active Directory

Start with Windows Server base installation. I used Windows Server 2012 R2 to test it. The same should likely work with earlier versions of Windows Server as well. ADFS 2.0 is part of Server 2012 R2, you might have to manually install it.

```
Install-WindowsFeature AD-Domain-Services, DNS -IncludeManagementTools

if ($env:userdomain -eq $env:computername)
{
   Install-ADDSForest –DomainName $domainname `
     -DomainNetbiosName $netbiosname `
     –DomainMode Win2012R2 –ForestMode `
     Win2012R2 –DatabasePath "C:\ADDATA\NTDS" `
     –SYSVOLPath "C:\ADDATA\SYSVOL" `
     –LogPath "C:\ADDATA\Logs" -Force `
     -SafeModeAdministratorPassword $password

  Restart-Computer
}
```

We will create a user “test”, this is the user account we will use to test SSO. We will also create two groups, AWS-Production and AWS-Dev and the user “test” is added to these groups. Everyone who is a member of AWS-Production will assume the role ADFS-Production in AWS, while the members of AWS-Dev will assume ADFS-Dev role. We will be creating these roles later. The user “test” is added to both AWS-Production and AWS-Dev groups. Because there is ambiguity, the user can assume either of the roles at AWS, it will prompt which role to assume. This highlights the generic case, if the user maps to a single role then the user is assumed that role without prompting. The following script creates the user/groups.

```
#Add new test user, that will used for SSO
New-ADUser -Name test -AccountPassword $password -EmailAddress "test@$domainname" -Enabled $true

#Create two groups and add test user to them
New-ADGroup AWS-Production -GroupScope Global  -GroupCategory Security
New-ADGroup AWS-Dev -GroupScope Global  -GroupCategory Security
Add-ADGroupMember -Identity AWS-Production -Members test
Add-ADGroupMember -Identity AWS-Dev -Members test
```

Install ADFS (Active Directory Federation Service)

ADFS provides simplified, secured identity federation and Web single sign-on (SSO) capabilities for end users who want to access applications within an ADFS-secured enterprise, in federation partner organizations, or in

the cloud.

In Windows Server 2012 R2, ADFS includes a federation role service that acts as an identity provider (authenticates users to provide security tokens to applications that trust ADFS) or as a federation provider (consumes tokens from other identity providers and then provides security tokens to applications that trust ADFS).

Script to install the ADFS role. This also generates a self-signed SSL certificate for https.

```
Install-WindowsFeature Web-Server, ADFS-Federation, Web-Scripting-Tools -IncludeManagementTools
Import-Module WebAdministration

#Add user account for ADFS service to run
New-ADUser -Name ADFSSVC -AccountPassword $password -Enabled $true
setspn -a host/localhost adfssvc

$adfsname = "adfs.$domainname"
#Generate Self signed Certificate

&$makecertpath -n "CN=$adfsname" -r -pe -sky exchange `
      -ss My -sr LocalMachine -eku 1.3.6.1.5.5.7.3.1
$sslcert = ,(Get-ChildItem 'Cert:\LocalMachine\My' `
      | Where-Object { $_.Subject -eq "CN=$adfsname" })[0]

$cred =  New-Object System.Management.Automation.PSCredential `
      ("$netbiosname\ADFSSVC", $password)
Install-AdfsFarm `
   -CertificateThumbprint $sslcert.Thumbprint `
   -FederationServiceDisplayName "Sivas ADFS" `
   -FederationServiceName $adfsname `
   -ServiceAccountCredential $cred `
   -OverwriteConfiguration
```

Establishing Two Way Trust

The enterprise ADFS server should trust Amazon and likewise AWS (your tenant) should trust your ADFS. Only then ADFS will send the claims and AWS will accept them. Command to configure ADFS to trust AWS is:

```
Add-ADFSRelyingPartyTrust -Name "Amazon" `
      -MetadataUrl https://signin.aws.amazon.com/static/saml-metadata.xml
```

Now that ADFS trusts AWS, we will establish the other side of the trust. For that you need to get the ADFS metadata and create an IAMS SAML provider (with this document). The ARN for the created SAML provider is saved in a variable. The account number is extracted from this ARN. These two variables are used later in the script.

```
#Download and save the ADFS metadata to a tempfile
#Because of self signed, need to disable the SSL Validation
#   otherwise WebClient.DownloadFile will fail.
$url = "https://localhost/FederationMetadata/2007-06/FederationMetadata.xml"
$metadatapath =  [IO.Path]::GetTempFileName()
[Net.ServicePointManager]::ServerCertificateValidationCallback = {$true}
$webClient = new-object System.Net.WebClient
$webClient.DownloadFile( $url, $metadatapath )

#Register a new SAML Provider with IAMS that has this ADFS information
$role = New-IAMSAMLProvider -Name "ADFS" -SAMLMetadataDocument (cat $metadatapath)
$account = $role.Substring(13,12)

del $metadatapath
```

Create AWS Role

You need to create one or more AWS roles. This is the role that will be assumed by matching the attributes of the incoming claim. In our case, we will be creating two roles, ADFS-Production and ADFS-Dev. As described, all users who are part of the AWS-Production group in the Active Directory can assume the ADFS-Production role. Similarly for ADFS-Dev, Each AWS role is associated with two policies. a) Trust policy that defines who can assume this role b) Access policy that defines what access the impersonated user has.

```
#custom replacement is used for $role, so don't have to deal with escape chars
$trustPolicy = @'
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRoleWithSAML",
      "Principal": {"Federated": "$role"},
      "Condition": {
        "StringEquals": {"SAML:aud": "https://signin.aws.amazon.com/saml"}
      }
    }
  ]
}
'@
$trustPolicy = $trustPolicy.Replace('$role', $role)

$accessPolicy = @'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": ["s3:Get*"],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
'@

New-IAMRole -AssumeRolePolicyDocument $trustPolicy -RoleName "ADFS-Production"
Write-IAMRolePolicy -RoleName "ADFS-Production" -PolicyName "EC2-Get-Access" -PolicyDocument $accessPolicy

New-IAMRole -AssumeRolePolicyDocument $trustPolicy -RoleName "ADFS-Dev"
Write-IAMRolePolicy -RoleName "ADFS-Dev" -PolicyName "EC2-Get-Access" -PolicyDocument $accessPolicy
```

Configure Assertions for the SAML Authentication Response

After a user's identity has been verified, ADFS must send an authentication response to the AWS endpoint (https://signin.aws.amazon.com/saml). This response must be a POST request that includes a SAML token that adheres to the HTTP POST Binding for SAML 2.0 standard, and that contains the following elements. All of these elements are required. You can find more info here.

Subject and NameID.: The following excerpt shows an example. Your own values would substitute for the marked ones. The value of the Recipient attribute inside the SubjectConfirmationData element must match the AWS endpoint (https://signin.aws.amazon.com/saml), as shown in the following example.

```
<Subject>
  <NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent">
            _cbb88bf52c2510eabe00c1642d4643f41430fe25e3
  </NameID>
  <SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
    <SubjectConfirmationData NotOnOrAfter="2013-11-05T02:06:42.876Z"
              Recipient="https://signin.aws.amazon.com/saml"/>
  </SubjectConfirmation>
</Subject>
```

PowerShell code fragment

```
$nameId = @'
    @RuleTemplate = "MapClaims"
    @RuleName = "Name ID"
    c:[Type == "http://schemas.microsoft.com/ws/2008/06/identity/claims/primarysid"]
       =>
    issue(Type = "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier",
       Issuer = c.Issuer,
       OriginalIssuer = c.OriginalIssuer,
       Value = c.Value,
       ValueType = c.ValueType,
       Properties["http://schemas.xmlsoap.org/ws/2005/05/identity/claimproperties/format"]
                              = "urn:oasis:names:tc:SAML:2.0:nameid-format:persistent");
'@
```

RoleSessionName: An attribute with Name set to https://aws.amazon.com/SAML/Attributes/RoleSessionName. The attribute value provides an identifier for the AWS temporary credentials that are issued for SSO and is used to display user information in the AWS console. This value must be between 2 and 32 characters long, can contain only alphanumeric characters, underscores, and the following characters: +=,.@-. It cannot contain spaces. For example,

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/RoleSessionName">
  <AttributeValue>name

</Attribute>
```

PowerShell code fragment:

```
$roleSessionName = @'
    @RuleTemplate = "LdapClaims"
    @RuleName = "RoleSessionName"
    c:[Type == "http://schemas.microsoft.com/ws/2008/06/identity/claims/windowsaccountname",
        Issuer == "AD AUTHORITY"]
        =>
    issue(store = "Active Directory",
        types = ("https://aws.amazon.com/SAML/Attributes/RoleSessionName"),
        query = ";mail;{0}", param = c.Value);
'@
```

Role. An attribute with Name set to https://aws.amazon.com/SAML/Attributes/Role. This attribute contains one or more AttributeValue elements that list the IAM role and SAML provider that the user is mapped to in your IdP. The role and provider are specified as a comma-delimited pair of ARNs, in the same format that they are used for the RoleArn and PrincipalArn parameters that are passed to AssumeRoleWithSAML. The attribute must contain at least one role/provider pair, and can contain multiple pairs. If the attribute contains multiple pairs, when the user uses WebSSO to sign into the AWS Management Console, he or she is asked to select the role to assume.

```
<Attribute Name="https://aws.amazon.com/SAML/Attributes/Role">
  <AttributeValue>arn:aws:iam::account-number:role/role-name,arn:aws:iam::account-number:saml-provider/provider-name

  <AttributeValue>arn:aws:iam::account-number:role/role-name,arn:aws:iam::account-number:saml-provider/provider-name

</Attribute>
```

PowerShell code fragment:

```
#list of AD groups is first stored in a temporary variables
$tempVariable = @'
    @RuleName = "Save AD Group Into http://temp/variable"
    c:[Type == "http://schemas.microsoft.com/ws/2008/06/identity/claims/windowsaccountname",
        Issuer == "AD AUTHORITY"]
        =>
    add(store = "Active Directory",
        types = ("http://temp/variable"),
        query = ";tokenGroups;{0}",
        param = c.Value);
'@

$roleMapping = @'
    @RuleName = "Role mapping"
    c:[Type == "http://temp/variable", Value =~ "(?i)^AWS-"]
        =>
    issue(Type = "https://aws.amazon.com/SAML/Attributes/Role", Value =
        RegExReplace(c.Value, "AWS-",
        "arn:aws:iam::$account:saml-provider/ADFS,arn:aws:iam::$account:role/ADFS-"));
'@
```

Now that the rules are stored in script variables, the following code will create the claim rules in ADFS.

```
$roleMapping = $roleMapping.Replace('$account', $account)
$ruleset = New-AdfsClaimRuleSet -ClaimRule $nameId,$roleSessionName, $tempVariable, $roleMapping

$issuanceAuthorizationRules = @'
    @RuleTemplate = "AllowAllAuthzRule"
        =>
    issue(Type = "http://schemas.microsoft.com/authorization/claims/permit",
        Value = "true");
'@
```

```
Set-ADFSRelyingPartyTrust -TargetName Amazon -IssuanceTransformRules $ruleset.ClaimRulesString -IssuanceAuthorizationRules
$issuanceAuthorizationRules
```

Testing

Now you are ready to do the testing! I had issues with IE, so I used Opera. Open Opera and invoke https://localhost/adfs/ls/IdpInitiatedSignOn.aspx from your ADFS server (you need to specify the ADFS server name, if trying remotely). Note: You need to use the user "test" to try this out. This is the user that was added to the appropriate groups.

References

1. Giving Console Access using SAML
2. Enabling Federation to AWS using Windows Active Directory, ADFS, and SAML 2.0

You can find the code under "AWS" folder at https://github.com/padisetty/Samples.

Explore & Enjoy!

/Siva

Posted by padisetty at 2:48 PM

Labels: AWS, PowerShell, Security

No comments:

Post a Comment

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)