Manage Azure AI Services Security

Security is a critical consideration for any application, and as a developer you should ensure that access to resources such as Azure AI services is restricted to only those who require it.

Access to Azure AI services is typically controlled through authentication keys, which are generated when you initially create an Azure AI services resource.

Clone the repository in Visual Studio Code

You'll develop your code using Visual Studio Code. The code files for your app have been provided in a GitHub repo.

> **Tip**: If you have already cloned the **mslearn-ai-services** repo recently, open it in Visual Studio code. Otherwise, follow these steps to clone it to your development environment.

1. Start Visual Studio Code.
2. Open the palette (SHIFT+CTRL+P) and run a **Git: Clone** command to clone the `https://github.com/MicrosoftLearning/mslearn-ai-services` repository to a local folder (it doesn't matter which folder).
3. When the repository has been cloned, open the folder in Visual Studio Code.
4. Wait while additional files are installed to support the C# code projects in the repo, if necessary

   > **Note**: If you are prompted to add required assets to build and debug, select **Not Now**.

5. Expand the `Labfiles/02-ai-services-security` folder.

Code for both C# and Python has been provided. Expand the folder of your preferred language.

Provision an Azure AI Services resource

If you don't already have one in your subscription, you'll need to provision an **Azure AI Services** resource.

1. Open the Azure portal at `https://portal.azure.com`, and sign in using the Microsoft account associated with your Azure subscription.
2. In the top search bar, search for *Azure AI services*, select **Azure AI Services**, and create an Azure AI services multi-service account resource with the following settings:
   - **Subscription**: *Your Azure subscription*
   - **Resource group**: *Choose or create a resource group (if you are using a restricted subscription, you may not have permission to create a new resource group - use the one provided)*
   - **Region**: *Choose any available region*
   - **Name**: *Enter a unique name*
   - **Pricing tier**: Standard S0
3. Select the required checkboxes and create the resource.
4. Wait for deployment to complete, and then view the deployment details.

Manage authentication keys

When you created your Azure AI services resource, two authentication keys were generated. You can manage these in the Azure portal or by using the Azure command line interface (CLI).

1. In the Azure portal, go to your Azure AI services resource and view its **Keys and Endpoint** page. This page contains the information that you will need to connect to your resource and use it from applications you develop. Specifically:

- An HTTP *endpoint* to which client applications can send requests.
- Two *keys* that can be used for authentication (client applications can use either of the keys. A common practice is to use one for development, and another for production. You can easily regenerate the development key after developers have finished their work to prevent continued access).
- The *location* where the resource is hosted. This is required for requests to some (but not all) APIs.

2. Now you can use the following command to get the list of Azure AI services keys, replacing *<resourceName>* with the name of your Azure AI services resource, and *<resourceGroup>* with the name of the resource group in which you created it.

   Code Copy

   ```
   az cognitiveservices account keys list --name <resourceName> --resource-group <resourceGroup>
   ```

   The command returns a list of the keys for your Azure AI services resource - there are two keys, named **key1** and **key2**.

   > **Tip**: If you haven't authenticated Azure CLI yet, run `az login` and sign into your account.

3. To test your Azure AI service, you can use **curl** - a command line tool for HTTP requests. In the **02-ai-services-security** folder, open **rest-test.cmd** and edit the **curl** command it contains (shown below), replacing *<yourEndpoint>* and *<yourKey>* with your endpoint URI and **Key1** key to use the Analyze Text API in your Azure AI services resource.

   Code Copy

   ```
   curl -X POST "<yourEndpoint>/language/:analyze-text?api-version=2023-04-01" -H "Content-Type: application/json" -H "Ocp-Apim-Subscription-Key: 81468b6728294aab99c489664a818197" --data-ascii "{'analysisInput':{'documents':[{'id':1,'text':'hello'}]}, 'kind': 'LanguageDetection'}"
   ```

4. Save your changes, and then run the following command:

   Code Copy

   ```
   ./rest-test.cmd
   ```

The command returns a JSON document containing information about the language detected in the input data (which should be English).

1. If a key becomes compromised, or the developers who have it no longer require access, you can regenerate it in the portal or by using the Azure CLI. Run the following command to regenerate your **key1** key (replacing *<resourceName>* and *<resourceGroup>* for your resource).

   Code Copy

   ```
   az cognitiveservices account keys regenerate --name <resourceName> --resource-group <resourceGroup> --key-name key1
   ```

The list of keys for your Azure AI services resource is returned - note that **key1** has changed since you last retrieved them.

1. Re-run the **rest-test** command with the old key (you can use the ^ arrow on your keyboard to cycle through previous commands), and verify that it now fails.

2. Edit the *curl* command in **rest-test.cmd** replacing the key with the new **key1** value, and save the changes. Then rerun the **rest-test** command and verify that it succeeds.

> **Tip**: In this exercise, you used the full names of Azure CLI parameters, such as **–resource-group**. You can also use shorter alternatives, such as **-g**, to make your commands less verbose (but a little harder to understand). The Azure AI Services CLI command reference lists the parameter options for each Azure AI services CLI command.

Secure key access with Azure Key Vault

You can develop applications that consume Azure AI services by using a key for authentication. However, this means that the application code must be able to obtain the key. One option is to store the key in an environment variable or a configuration file where the application is deployed, but this approach leaves the key vulnerable to unauthorized access. A better approach when developing applications on Azure is to store the key securely in Azure Key Vault, and provide access to the key through a *managed identity* (in other words, a user account used by the application itself).

Create a key vault and add a secret

First, you need to create a key vault and add a *secret* for the Azure AI services key.

1. Make a note of the **key1** value for your Azure AI services resource (or copy it to the clipboard).
2. In the Azure portal, on the **Home** page, select the **+ Create a resource** button, search for *Key Vault*, and create a **Key Vault** resource with the following settings:
   - **Basics** tab
     - **Subscription**: *Your Azure subscription*
     - **Resource group**: *The same resource group as your Azure AI service resource*
     - **Key vault name**: *Enter a unique name*
     - **Region**: *The same region as your Azure AI service resource*
     - **Pricing tier**: Standard
   - **Access configuration** tab
     - **Permission model**: Vault access policy
     - Scroll down to the **Access policies** section and select your user using the checkbox on the left. Then select **Review + create**, and select **Create** to create your resource.
3. Wait for deployment to complete and then go to your key vault resource.
4. In the left navigation pane, select **Secrets** (in the Objects section).
5. Select **+ Generate/Import** and add a new secret with the following settings :
   - **Upload options**: Manual
   - **Name**: AI-Services-Key *(it's important to match this exactly, because later you'll run code that retrieves the secret based on this name)*
   - **Value**: *Your **key1** Azure AI services key*
6. Select **Create**.

Create a service principal

To access the secret in the key vault, your application must use a service principal that has access to the secret. You'll use the Azure command line interface (CLI) to create the service principal, find its object ID, and grant access to the secret in Azure Vault.

1. Run the following Azure CLI command, replacing *<spName>* with a unique suitable name for an application identity (for example, *ai-app* with your initials appended on the end; the name must be unique within your tenant). Also replace *<subscriptionId>* and *<resourceGroup>* with the correct values for your subscription ID and the resource group containing your Azure AI services and key vault resources:

> **Tip**: If you are unsure of your subscription ID, use the **az account show** command to retrieve your subscription information - the subscription ID is the **id** attribute in the output. If you see an error about the object already existing, please choose a different unique name.

Code Copy

```
az ad sp create-for-rbac -n "api://<spName>" --role owner --scopes
subscriptions/<subscriptionId>/resourceGroups/<resourceGroup>
```

The output of this command includes information about your new service principal. It should look similar to this:

Code Copy

```
{
    "appId": "abcd12345efghi67890jklmn",
    "displayName": "api://ai-app-",
    "password": "1a2b3c4d5e6f7g8h9i0j",
    "tenant": "1234abcd5678fghi90jklm"
}
```

Make a note of the **appId**, **password**, and **tenant** values - you will need them later (if you close this terminal, you won't be able to retrieve the password; so it's important to note the values now - you can paste the output into a new text file on your local machine to ensure you can find the values you need later!)

1. To get the **object ID** of your service principal, run the following Azure CLI command, replacing *<appId>* with the value of your service principal's app ID.

   Code Copy

   ```
   az ad sp show --id <appId>
   ```

2. Copy the `id` value in the json returned in response.

3. To assign permission for your new service principal to access secrets in your Key Vault, run the following Azure CLI command, replacing *<keyVaultName>* with the name of your Azure Key Vault resource and *<objectId>* with the value of your service principal's ID value you've just copied.

   Code Copy

   ```
   az keyvault set-policy -n <keyVaultName> --object-id <objectId> --secret-permissions get
   list
   ```

Use the service principal in an application

Now you're ready to use the service principal identity in an application, so it can access the secret Azure AI services key in your key vault and use it to connect to your Azure AI services resource.

> **Note**: In this exercise, we'll store the service principal credentials in the application configuration and use them to authenticate a **ClientSecretCredential** identity in your application code. This is fine for development and testing, but in a real production application, an administrator would assign a *managed identity* to the application so that it uses the service principal identity to access resources, without caching or storing the password.

1. In your terminal, switch to the **C-Sharp** or **Python** folder depending on your language preference by running `cd C-Sharp` or `cd Python`. Then run `cd keyvault_client` to navigate to the app folder.

2. Install the packages you will need to use for Azure Key Vault and the Text Analytics API in your Azure AI services resource by running the appropriate command for your language preference:

   C#

   Code Copy

   ```
   dotnet add package Azure.AI.TextAnalytics --version 5.3.0
   dotnet add package Azure.Identity --version 1.5.0
   dotnet add package Azure.Security.KeyVault.Secrets --version 4.2.0-beta.3
   ```

   Python

   Code Copy

   ```
   pip install azure-ai-textanalytics==5.3.0
   pip install azure-identity==1.5.0
   pip install azure-keyvault-secrets==4.2.0
   ```

3. View the contents of the **keyvault-client** folder, and note that it contains a file for configuration settings:

   - **C#**: appsettings.json
   - **Python**: .env

   Open the configuration file and update the configuration values it contains to reflect the following settings:

   - The **endpoint** for your Azure AI Services resource
   - The name of your **Azure Key Vault** resource
   - The **tenant** for your service principal
   - The **appId** for your service principal
   - The **password** for your service principal

   Save your changes by pressing **CTRL+S**.

4. Note that the **keyvault-client** folder contains a code file for the client application:

   - **C#**: Program.cs
   - **Python**: keyvault-client.py

   Open the code file and review the code it contains, noting the following details:

   - The namespace for the SDK you installed is imported
   - Code in the **Main** function retrieves the application configuration settings, and then it uses the service principal credentials to get the Azure AI services key from the key vault.
   - The **GetLanguage** function uses the SDK to create a client for the service, and then uses the client to detect the language of the text that was entered.

5. Enter the following command to run the program:

   C#

   Code Copy

   ```
   dotnet run
   ```

   Python

   Code Copy

```
python keyvault-client.py
```

6. When prompted, enter some text and review the language that is detected by the service. For example, try entering “Hello”, “Bonjour”, and “Gracias”.
7. When you have finished testing the application, enter “quit” to stop the program.

Clean up resources

If you’re not using the Azure resources created in this lab for other training modules, you can delete them to avoid incurring further charges.

1. Open the Azure portal at `https://portal.azure.com`, and in the top search bar, search for the resources you created in this lab.

2. On the resource page, select **Delete** and follow the instructions to delete the resource. Alternatively, you can delete the entire resource group to clean up all resources at the same time.

More information

For more information about securing Azure AI services, see the Azure AI Services security documentation.