Integrating Django Authentication with Microsoft Entra ID using SAML on IIS

This report provides a comprehensive, step-by-step tutorial for integrating SAML 2.0 based Single Sign-On (SSO) between a Django web application hosted on Internet Information Services (IIS) within an Azure Virtual Machine (VM) and Microsoft Entra ID (formerly Azure Active Directory) acting as the Identity Provider (IdP). This guide is designed for beginners in SAML integration.

Introduction

Integrating enterprise applications with a centralized identity management system offers significant benefits, including enhanced security, improved user experience through Single Sign-On (SSO), and simplified administration. Microsoft Entra ID is a widely used cloud-based identity and access management service. Security Assertion Markup Language (SAML) 2.0 is an open standard protocol that enables secure exchange of authentication and authorization data between an Identity Provider (like Entra ID) and a Service Provider (like your Django application).

This tutorial will guide you through the process of:

1. Understanding the core concepts of SAML 2.0.
2. Registering and configuring your Django application within Microsoft Entra ID.
3. Integrating a Python SAML library into your Django project.
4. Configuring IIS to correctly handle SAML flows for your Django application.
5. Implementing authorization based on information received from Entra ID.
6. Troubleshooting common integration issues.

By following these steps, you can leverage Entra ID for secure and seamless user authentication in your Django application deployed on an Azure VM using IIS.

Step 1: Understanding SAML 2.0 Fundamentals

Before diving into the implementation, it's essential to understand the basic concepts of SAML 2.0. SAML is an XML-based standard that allows security domains to exchange user authentication and authorization data.

- **Identity Provider (IdP):** The entity responsible for authenticating the user and providing identity information. In this scenario, Microsoft Entra ID is the IdP. The IdP maintains the user directory and credentials.
- **Service Provider (SP):** The application or service that the user wants to access. The SP relies on the IdP to authenticate the user. Here, your Django application hosted on IIS is the SP.
- **Principal (Subject):** Typically, the human user attempting to access the SP.
- **SAML Assertion:** An XML document generated by the IdP after successful user authentication. It contains statements (assertions) about the user, such as their identity (e.g., email address or username), attributes (e.g., name, group memberships), and authentication context (how they authenticated). The SP uses this assertion to grant

access. Think of it like a temporary digital ID card issued by the IdP.

- **SAML Bindings:** Define how SAML messages (like authentication requests and responses) are transported between the IdP and SP using standard communication protocols, typically HTTP. Common bindings include:
  - **HTTP-Redirect:** Often used for sending SAML Authentication Requests (AuthnRequest) from the SP to the IdP. The request is encoded and included as a query parameter in the URL.
  - **HTTP-POST:** Often used for sending SAML Responses from the IdP back to the SP's Assertion Consumer Service (ACS). The XML response is typically Base64 encoded and embedded within an HTML form that auto-submits via JavaScript.
  - **HTTP-Artifact:** An older binding where the IdP sends a small reference (artifact) to the SP via the browser, and the SP then communicates directly with the IdP (back-channel) to retrieve the full assertion using the artifact. This is less common now due to its complexity.
- **Metadata:** An XML document that describes the configuration of either an IdP or an SP.
  - **IdP Metadata:** Contains information about the IdP, such as its entity ID, SSO and Single Logout (SLO) endpoint URLs, supported bindings, and its public signing certificate. The SP uses this to know where to send requests and how to verify responses. Entra ID provides a metadata URL for each application.
  - **SP Metadata:** Contains information about the SP, such as its entity ID, Assertion Consumer Service (ACS) URL, SLO endpoint URL, and potentially its public certificate for signing requests or decrypting assertions. The IdP uses this to know where to send responses and verify requests. Your Django SAML library will typically provide an endpoint to expose this metadata.

The core idea is that the user authenticates *only* with the IdP (Entra ID). The IdP then provides a digitally signed assertion to the SP (Django), vouching for the user's identity and attributes. The SP trusts the IdP (configured via metadata exchange) and uses the assertion to log the user in without needing their password. This enables SSO – log in once to the IdP, access multiple SPs.

Step 2: Registering the Entra ID Enterprise Application

To use Entra ID as your IdP, you need to register your Django application as an "Enterprise Application" within your Entra ID tenant. This registration process establishes the trust relationship and configures how Entra ID interacts with your application via SAML.

Prerequisites:

- An Azure account with an active subscription and an Entra ID tenant.
- Appropriate administrative permissions in Entra ID (e.g., Cloud Application Administrator, Application Administrator).

Steps:

1. **Navigate to Enterprise Applications:**
   - Sign in to the Microsoft Entra admin center.
   - Browse to **Identity** > **Applications** > **Enterprise applications**.
2. **Create a New Application:**
   - Click **+ New application**.
   - Click **+ Create your own application**.
   - Enter a descriptive name for your application (e.g., "My Django SAML App").

   - Select the option **Integrate any other application you don't find in the gallery (Non-gallery)**. *Do not select a pre-existing gallery application unless it specifically matches your exact setup and requirements.* Non-gallery allows for full custom SAML configuration.
   - Click **Create**.
3. **Configure Single Sign-On (SSO):**
   - Once the application overview page loads, navigate to **Single sign-on** in the left-hand menu under "Manage".
   - Select **SAML** as the single sign-on method. This will open the "Set up Single Sign-On with SAML" configuration page.
4. **Configure Basic SAML Settings:**
   - Locate the **Basic SAML Configuration** section (usually Box 1) and click **Edit**.
   - You will need two crucial URLs from your Django application (which you will configure in the next step using your chosen SAML library). For now, you might need placeholder values, or you can come back to this step after configuring Django. The required fields are:
     - **Identifier (Entity ID):** This is a globally unique URI that identifies your Django application (the SP). It must exactly match the entityId configured in your Django SAML settings. It's often recommended to use the URL where your SP's metadata will be hosted. Example format: https://your-django-app.com/saml/metadata/. You might need to add multiple identifiers if required by your setup.
     - **Reply URL (Assertion Consumer Service URL):** This is the endpoint on your Django application where Entra ID will send the SAML Response (assertion) after successful authentication. It must exactly match the Assertion Consumer Service (ACS) URL configured in your Django SAML library. Example format: https://your-django-app.com/saml/acs/. Ensure the protocol (HTTPS is strongly recommended) and path match precisely.
     - **Sign on URL (Optional):** If you want to support IdP-initiated SSO (where users start from Entra ID's "My Apps" portal), enter the URL users should land on in your application. For SP-initiated flows (starting from your Django app), this is often left blank or set to the application's homepage.
     - **Relay State (Optional):** Used to pass state information through the IdP during the SSO flow. Can be useful for redirecting users to a specific page after login. Often left blank initially.
     - **Logout Url (Optional):** The endpoint on your application where Entra ID can send SAML Logout Requests or Responses. Example format: https://your-django-app.com/saml/sls/.
   - **Important:** The values for Identifier and Reply URL *must* exactly match what your Django application expects. Mismatches are a common source of errors. Ensure consistency in scheme (http/https), domain, path, and trailing slashes.
   - Click **Save**.
5. **Define User Attributes & Claims:**
   - Locate the **Attributes & Claims** section (usually Box 2) and click **Edit**.
   - Claims are pieces of information about the user that Entra ID includes in the SAML assertion. Your Django application will use these claims to identify the user and potentially determine their permissions.
   - **Unique User Identifier (NameID):** This is the primary identifier for the user. By

default, Entra ID often uses user.userprincipalname (UPN). You might need to change this depending on how you identify users in Django. Clicking on the "Unique User Identifier" claim allows you to change the source attribute (e.g., user.mail, user.objectid) and the NameID format. For integration with systems synchronized from on-premises AD, using an immutable identifier like objectid or a transformed on-premises attribute might be more reliable than UPN or email, which can change. urn:oasis:names:tc:SAML:2.0:nameid-format:persistent is often a good choice for a stable, unique identifier.

- **Additional Claims:** Entra ID includes some default claims (like email, first name, last name). You need to ensure the claims your Django application expects are present and correctly named.
    - Commonly needed claims and their typical Entra ID source attributes :
        - http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress -> user.mail (or just name it email)
        - http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname -> user.givenname (or just name it firstName or first_name)
        - http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname -> user.surname (or just name it lastName or last_name)
    - You can **Add new claim** or edit existing ones. Ensure the **Name** of the claim matches what your Django SAML library expects (check the library's attribute mapping configuration). The **Namespace** is often left blank unless a specific URI format is required.
- **Adding Group Claims:** This is crucial for authorization based on Entra ID group membership.
    - Click **+ Add a group claim**.
    - Choose which groups to include (e.g., **Security groups**, **All groups**, or **Groups assigned to the application**). "Groups assigned to the application" is often preferred for better control.
    - Select the **Source attribute**. For authorization purposes, **Group ID** is strongly recommended as it's unique and immutable. Other options like sAMAccountName or Cloud-only group display names exist but might lead to collisions or issues if names change.
    - Under **Advanced options**, you can **Customize the name of the group claim**. Set the **Name** to something simple like groups (this is the claim name your Django app will look for). Ensure **Emit group name for cloud-only groups** is checked if needed.
    - Click **Save**.
- **Important Note on Group Limits:** Entra ID limits the number of group IDs it sends in a SAML token (typically 150-200). If a user is in more groups than the limit, the group claim might be incomplete or replaced with a link to query the Graph API. This needs consideration when designing authorization logic. Using "Groups assigned to the application" can help manage this limit.

6. **Assign Users and Groups:**
    - Navigate to **Users and groups** in the left-hand menu under "Manage".
    - Click **+ Add user/group**.
    - Select the users or security groups that should be allowed to sign in to your Django application via this SAML configuration. Start with just your test user account initially.

    - Click **Assign**.
    - Alternatively, you can go to **Properties** under "Manage" and set **Assignment required?** to **No**. This allows any user in the tenant to attempt login (they still need to pass authentication and any Conditional Access policies), which might be suitable for self-service scenarios but offers less control. For initial testing and most enterprise scenarios, requiring assignment (Yes) is recommended.
7. **Obtain IdP Metadata and Certificate:**
    - Navigate back to the **Single sign-on** page for your application.
    - Locate the **SAML Certificates** section (usually Box 3).
    - **App Federation Metadata Url:** Find this URL. It points to the XML metadata document for this specific application configuration within your Entra ID tenant. Copy this URL; you will need it for your Django settings. The format is typically https://login.microsoftonline.com/{TenantID}/federationmetadata/2007-06/federationmetadata.xml?appid={AppID}.
    - **Certificate (Base64):** Find the active signing certificate. Click the **Download** link next to **Certificate (Base64)**. This will download a .cer file containing the public key certificate that Entra ID uses to sign the SAML assertions. You will need the content of this certificate (the Base64 encoded string) for your Django configuration to verify the responses from Entra ID. You can open the .cer file in a text editor to copy the content. Ensure you copy the entire content, often including the -----BEGIN CERTIFICATE----- and -----END CERTIFICATE----- markers if required by your library.
    - **Federation Metadata XML (Alternative):** You can also download the entire metadata as an XML file. Some libraries allow configuring the IdP by pointing to this local file instead of the URL and certificate string separately.
    - **Other URLs:** Note down the **Login URL** and **Microsoft Entra Identifier** (which is the IdP's Entity ID, e.g., https://sts.windows.net/{TenantID}/) from the **Set up <YourAppName>** section (usually Box 4). These might be needed for manual configuration if you don't use the metadata URL.

You have now successfully registered your Django application in Entra ID and configured the basic SAML settings, claims, user assignments, and obtained the necessary information (Metadata URL, Certificate) to configure the Service Provider side in Django.

Step 3: Setting Up the Django SAML Service Provider

Now, you need to configure your Django application to act as a SAML Service Provider (SP) that communicates with the Entra ID IdP you just set up. This involves choosing and installing a Python SAML library and configuring it within your Django project.

3.1. Choosing and Installing a Python SAML Library

Several Python libraries can help implement SAML SP functionality in Django. Two prominent options are:

1. **python3-saml (and wrappers like python3-saml-django):**
    - **Description:** A widely used toolkit specifically focused on implementing the SP side of SAML 2.0. It's part of a family of SAML toolkits (PHP, Java, Ruby, etc.) originally developed by OneLogin, making its structure potentially familiar if you've used other

    toolkits. It handles SSO, SLO, assertion/NameID encryption, and message signing. It relies on the external xmlsec library for cryptographic operations.
   - **Django Integration:** While python3-saml itself is framework-agnostic, libraries like python3-saml-django provide convenient wrappers, including Django authentication backends, URL patterns, and settings integration, simplifying the setup considerably.
   - **Pros:** Focused SP implementation, active (part of SAML-Toolkits org), good documentation and examples (including Django demos) , potentially easier configuration settings compared to pysaml2.
   - **Cons:** Requires the xmlsec1 binary to be installed on the system, which can sometimes cause installation issues, especially on Windows or specific environments.
2. **djangosaml2 (based on pysaml2):**
   - **Description:** A Django application that integrates the pysaml2 library. pysaml2 is a more comprehensive library that can implement *both* SP and IdP roles. djangosaml2 provides features like various bindings, discovery service support, IdP hinting/scoping, and an authentication backend.
   - **Pros:** Mature library (pysaml2 is older), supports both SP and IdP roles (though only SP is needed here), offers advanced features like discovery service.
   - **Cons:** Also requires xmlsec1 binary. Configuration can be perceived as more complex than python3-saml. Historically, there have been concerns about compatibility with newer Django versions, although recent versions seem to address this. Some users report difficulties getting examples working compared to python3-saml.

Recommendation: For a beginner focused on implementing only the SP side with Entra ID, using **python3-saml via the python3-saml-django wrapper** is often a good starting point due to its focused nature and potentially simpler configuration. This tutorial will proceed using python3-saml-django.
Installation:

1. **Install xmlsec1:** This is a system dependency. The method varies by OS:
   - **Debian/Ubuntu:** sudo apt-get update && sudo apt-get install libxml2-dev libxmlsec1-dev libxmlsec1-openssl
   - **RHEL/CentOS/Fedora:** sudo yum install libxml2-devel xmlsec1-devel xmlsec1-openssl-devel libtool-ltdl-devel
   - **Windows:** Installation can be trickier. Refer to python-xmlsec documentation or potential pre-built wheels. Ensure the xmlsec1 executable is in the system's PATH.
   - **Alpine Linux:** apk add --no-cache xmlsec-dev xmlsec-openssl-dev libxml2-dev libxslt-dev
2. **Install the Python Package:** Activate your Django project's virtual environment and install the wrapper library using pip. This will also install python3-saml as a dependency.
   ```
   pip install python3-saml-django
   ```

3.2. Configuring settings.py

You need to add the SAML configuration to your Django project's settings.py. python3-saml-django simplifies this by allowing you to define settings directly in settings.py.

1. **Add to INSTALLED_APPS and AUTHENTICATION_BACKENDS:**

```
# settings.py
INSTALLED_APPS = [
    #... other apps
    'django.contrib.auth',
    'django.contrib.sessions',
    #...
    'django_saml', # Add this line
]

AUTHENTICATION_BACKENDS =
```

The SamlUserBackend handles user authentication based on the SAML assertion. Keeping ModelBackend allows standard Django username/password login (e.g., for the admin site or fallback).

2. **Define SAML SP Configuration (SAML_SP):** This dictionary defines your Django application (Service Provider).

```
# settings.py
import os
from django.urls import reverse_lazy

BASE_DIR =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
# Define the base URL of your application dynamically if possible,
or hardcode for testing
# Ensure this matches exactly what's in Entra ID (including
https://)
MY_APP_BASE_URL = 'https://your-azure-vm-dns-or-ip' # CHANGE THIS

SAML_SP = {
    # Matches 'Identifier (Entity ID)' in Entra ID
    "entityId": f"{MY_APP_BASE_URL}/saml/metadata/",
    "assertionConsumerService": {
        # Matches 'Reply URL (Assertion Consumer Service URL)' in
Entra ID
        "url": f"{MY_APP_BASE_URL}/saml/acs/",
        "binding":
"urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
    },
    "singleLogoutService": {
        # Matches 'Logout Url' in Entra ID (optional but
recommended)
        "url": f"{MY_APP_BASE_URL}/saml/sls/",
        "binding":
"urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
    },
    # Specifies the format for the NameID requested from the IdP
    "NameIDFormat":
"urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified",
    # Optional: If signing requests or decrypting assertions,
provide paths to cert/key
    # "x509cert": os.path.join(BASE_DIR, 'saml', 'certs',
'sp.crt'),
    # "privateKey": os.path.join(BASE_DIR, 'saml', 'certs',
'sp.key'),
}
```

- **Crucially**, ensure entityId and assertionConsumerService.url exactly match the **Identifier (Entity ID)** and **Reply URL (ACS URL)** you configured in Entra ID (Step 2.4). Mismatches will cause errors. Use HTTPS if your application is served over HTTPS.
- The NameIDFormat tells Entra ID what kind of user identifier your application prefers.
- The x509cert and privateKey are only needed if you enable signing SAML requests (authnRequestsSigned in SAML_SECURITY) or if Entra ID is configured to encrypt assertions sent to your SP. For basic SSO, they might not be required initially. If used, ensure the saml/certs/ directory exists and contains the files.

3. **Define SAML IdP Configuration (Choose ONE method):** You need to tell Django about Entra ID.
   - **Method A: Using Metadata URL (Recommended):** This is the simplest and most dynamic way. The library will fetch and cache the IdP configuration.
     ```
     # settings.py
     SAML_IDP_URL = 'PASTE_YOUR_APP_FEDERATION_METADATA_URL_HERE'
     # Optional: Timeout for caching the metadata (in seconds)
     SAML_IDP_METADATA_TIMEOUT = 3600 # 1 hour (default)
     ```
     Replace the placeholder with the **App Federation Metadata Url** obtained in Step 2.7.
   - **Method B: Manual Configuration (SAML_IDP dictionary):** If the metadata URL is unavailable or you prefer explicit configuration.
     ```
     # settings.py
     SAML_IDP = {
         # Matches 'Microsoft Entra Identifier' from Entra ID
     setup (Box 4)
         "entityId": "https://sts.windows.net/YOUR_TENANT_ID/",
         "singleSignOnService": {
             # Matches 'Login URL' from Entra ID setup (Box 4)
             "url":
     "https://login.microsoftonline.com/YOUR_TENANT_ID/saml2",
             "binding":
     "urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
         },
         "singleLogoutService": {
             # Matches 'Logout URL' from Entra ID setup (Box 4)
             "url":
     "https://login.microsoftonline.com/YOUR_TENANT_ID/saml2",
             "binding":
     "urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
         },
     ```

```
    # Paste the Base64 certificate content obtained in Step
2.7
    # Remove -----BEGIN CERTIFICATE----- and -----END
CERTIFICATE----- lines
    # Ensure it's a single string without line breaks
    "x509cert": "PASTE_IDP_BASE64_CERT_CONTENT_HERE"
}
```

Replace placeholders with your Tenant ID and the certificate content.

- **Method C: Using Local Metadata File:**

  ```
  # settings.py
  SAML_IDP_METADATA_FILE = os.path.join(BASE_DIR, 'saml',
  'idp_metadata.xml')
  ```

  Place the downloaded Federation Metadata XML file (from Step 2.7) into saml/idp_metadata.xml.

4. **Configure Attribute Mapping (SAML_ATTR_MAP):** Map the SAML assertion attribute names (from Entra ID claims, Step 2.5) to your Django User model fields.

```
# settings.py
# Keys are SAML attribute names (check Entra ID claims config or
SAML tracer)
# Values are Django User model field names
SAML_ATTR_MAP =

# Attribute to use for identifying the user in Django (must be
unique)
# Ensure this corresponds to a unique value from the SAML
assertion (e.g., NameID or email)
SAML_USERNAME_ATTR = 'email' # Or potentially 'NameID' if using
NameID mapping

# Optional: Default values if an attribute is missing in the
assertion
SAML_ATTR_DEFAULTS = {
    'first_name': '',
    'last_name': '',
}

# Optional: Attributes to ignore on subsequent logins if
SAML_UPDATE_USER is True
SAML_ATTR_UPDATE_IGNORE = ['email']
```

- The keys in SAML_ATTR_MAP **must match the claim names** sent by Entra ID. Use a SAML tracer (Step 7.2) to confirm the exact names if unsure. Full URIs are often used.
- SAML_USERNAME_ATTR tells the backend which mapped attribute holds the unique identifier to find/create the Django user. This *must* correspond to a unique user identifier from the assertion.
- Mapping the groups attribute directly often requires custom logic in the backend or signals, as it usually contains a list of group IDs, not directly mappable to Django's

user.groups ManyToMany field.

5. **Configure Other Settings:**

```
# settings.py

# Where to redirect after successful login if no 'next' parameter
is provided
LOGIN_REDIRECT_URL = '/' # Or reverse_lazy('some_named_url')
# Where to redirect after logout
SAML_LOGOUT_REDIRECT = '/logged-out/' # A simple page indicating
logout success

# Set to True in production for security
SAML_STRICT = True # Default is True
# Set to False in production
SAML_DEBUG = False # Default is False

# Allow creating new Django users if they don't exist upon SAML
login
SAML_CREATE_USER = True # Default is True
# Update existing user attributes based on SAML data on each login
SAML_UPDATE_USER = False # Default is False

# Optional: Advanced security settings
SAML_SECURITY = {
    "authnRequestsSigned": False, # Set to True if SP signs
requests
    "logoutRequestSigned": False, # Set to True if SP signs logout
requests
    "logoutResponseSigned": False, # Set to True if SP signs
logout responses
    "wantMessagesSigned": False, # Set to True if IdP must sign
SAML Response/Logout
    "wantAssertionsSigned": True, # Recommended: Set to True if
IdP must sign Assertions
    "wantNameId": True, # Default: Require NameID from IdP
    # Add other security flags as needed (e.g., encryption,
signature algorithms)
    # "signatureAlgorithm":
"http://www.w3.org/2001/04/xmldsig-more#rsa-sha256",
    # "digestAlgorithm":
"http://www.w3.org/2001/04/xmlenc#sha256",
}

# Optional: Define path for SP certs if signing/encryption is
enabled
# SAML_BASE_DIRECTORY = os.path.join(BASE_DIR, 'saml')

# Optional: Useful behind load balancers/proxies if URL detection
```

```
is wrong
# SAML_DESTINATION_HOST = 'your-public-domain.com'
# SAML_DESTINATION_HTTPS = True # If public access is HTTPS
```

- Review the SAML_SECURITY settings carefully. For Entra ID, assertions are typically signed, so wantAssertionsSigned: True is recommended. Check Entra ID's configuration for message signing requirements.
- SAML_CREATE_USER controls whether users authenticated via SAML but not existing in Django are automatically created.
- SAML_UPDATE_USER controls if first_name, last_name, etc., are updated from SAML attributes on every login.

3.3. Configuring urls.py

You need to include the URL patterns provided by django_saml to handle the SAML endpoints.

1. **Include django_saml.urls:** In your project's main urls.py (or an app's urls.py if organizing differently), include the library's URLs under a specific path prefix (e.g., saml/).
   ```
   # your_project/urls.py
   from django.contrib import admin
   from django.urls import path, include
   from django.views.generic import TemplateView # For logout page
   example

   urlpatterns =
   ```
2. **Understanding the Endpoints:** Including django_saml.urls under the /saml/ prefix automatically creates these key endpoints:
   - /saml/login/: (Default name: saml_login) Initiates the SP-initiated SSO flow. You typically link to this URL from your application's login button/page.
   - /saml/acs/: (Default name: saml_acs) The Assertion Consumer Service. This URL **must match** the assertionConsumerService.url in SAML_SP and the Reply URL in Entra ID. It handles the incoming SAML Response.
   - /saml/sls/: (Default name: saml_sls) The Single Logout Service. Handles incoming Logout Requests/Responses.
   - /saml/metadata/: (Default name: saml_metadata) Serves the SP's metadata XML. This URL **must match** the entityId in SAML_SP and the Identifier (Entity ID) in Entra ID.

Ensure the base path (saml/ in this example) used in urls.py is consistent with the paths defined in your SAML_SP configuration in settings.py and the corresponding URLs (Identifier, Reply URL, Logout URL) configured in Entra ID.

Step 4: Configuring IIS for SAML

Hosting a Django application on IIS requires a bridge between IIS and Python's Web Server Gateway Interface (WSGI). Additionally, specific configurations might be needed to ensure SAML redirects and callbacks work correctly, especially when operating behind proxies or load balancers common in Azure deployments.

4.1. Choosing a Handler (wfastcgi vs. HttpPlatformHandler)

Two common methods connect IIS to Python/Django:

- **wfastcgi:** A library providing a bridge using the FastCGI protocol. It requires configuring FastCGI settings in IIS and adding a handler mapping in web.config.
- **HttpPlatformHandler:** A more recent IIS module designed to manage and proxy requests to various HTTP listeners, including Python WSGI servers like Waitress, Gunicorn, or Uvicorn.

Microsoft no longer actively maintains wfastcgi and recommends using HttpPlatformHandler for hosting Python applications on IIS. Therefore, this tutorial focuses on HttpPlatformHandler.
Installing HttpPlatformHandler:

1. Download the appropriate installer (x86 or x64) for your Windows Server version from the official Microsoft sources (search for "IIS HttpPlatformHandler download").
2. Run the installer on your Azure VM where IIS is installed.
3. Restart IIS (e.g., using iisreset in an administrative command prompt) might be necessary.

4.2. web.config Configuration for HttpPlatformHandler

Create a web.config file in the root directory of your Django project (where manage.py resides). This file tells IIS how to use HttpPlatformHandler to run your Django application.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.webServer>
    <handlers>
      <add name="PythonHttpPlatformHandler" path="*" verb="*"
modules="httpPlatformHandler" resourceType="Unspecified"
requireAccess="Script" />
    </handlers>
    <httpPlatform
processPath="C:\path\to\your\venv\Scripts\python.exe" arguments="-m
waitress --port %HTTP_PLATFORM_PORT%
your_project_name.wsgi:application" stdoutLogEnabled="true"
                  stdoutLogFile=".\logs\python_stdout.log"
startupTimeLimit="60"
                  processesPerApplication="1">
      <environmentVariables>
        <environmentVariable name="DJANGO_SETTINGS_MODULE"
value="your_project_name.settings" /> </environmentVariables>
    </httpPlatform>
  </system.webServer>
</configuration>
```

Explanation:

- **<handlers>:** Registers httpPlatformHandler to handle all incoming requests (path="*", verb="*").
- **<httpPlatform>:** Configures the handler:

- processPath: **Crucially**, point this to the python.exe inside your project's **virtual environment**. Do not use the system-wide Python installation unless absolutely necessary.
- arguments: This specifies the command to start your Python WSGI server.
    - **Do NOT use manage.py runserver**. It's a development server, unsuitable and insecure for production.
    - Use a production-ready WSGI server like waitress (pure Python, easy on Windows), gunicorn (common on Linux, might work on Windows via WSL or with limitations), or uvicorn (for ASGI applications).
    - The example uses waitress-serve. Ensure waitress is installed in your virtual environment (pip install waitress).
    - Replace your_project_name.wsgi:application with the actual path to your WSGI application object.
    - --port %HTTP_PLATFORM_PORT%: This is essential. HttpPlatformHandler assigns a dynamic port for the Python process to listen on and passes it via the HTTP_PLATFORM_PORT environment variable. Your WSGI server command *must* use this variable to listen on the correct port.
- stdoutLogEnabled="true" / stdoutLogFile: **Essential for debugging**. Any output (including errors) from the Python process startup will be logged here. Check this file first if your site fails to load (e.g., 502.3 Bad Gateway errors). Ensure the specified directory exists and the IIS Application Pool identity has write permissions to it.
- startupTimeLimit: Time IIS waits for the Python process to start listening on the port. Increase if your app takes longer to initialize.
- environmentVariables: Sets environment variables for the Python process.
    - DJANGO_SETTINGS_MODULE: Required for Django to find your settings.
    - PYTHONPATH: Sometimes needed if your project structure requires adding the root directory to the Python path.
    - HTTPS: May be needed if using URL Rewrite to signal HTTPS (see below).

4.3. Handling Proxies and Headers (URL Rewrite)

A significant challenge arises when IIS is deployed behind a reverse proxy, load balancer, or Application Gateway in Azure. These components often terminate the external HTTPS connection and forward the request to the IIS server internally, possibly over HTTP and using an internal hostname or IP address.
This mismatch causes problems for SAML because:

1. The SAML library needs to know the application's public-facing URL (e.g., https://your-public-domain.com/saml/acs/) to validate the Destination attribute in the incoming SAML Response from Entra ID.
2. The library needs the correct public URL to generate the SP metadata with accurate endpoint URLs.

If IIS/Django only sees the internal HTTP request, the SAML validation will likely fail because the Destination in the SAML response (https://your-public-domain.com/...) won't match the URL the application *thinks* it's serving (http://internal-ip/...).
Proxies typically add headers like X-Forwarded-Proto (value https) and X-Forwarded-Host (value your-public-domain.com) to the request forwarded to IIS. The IIS URL Rewrite Module can be used to read these headers and adjust the request information passed to Django or set

environment variables that the SAML library can use.
Example URL Rewrite Rule (add within <system.webServer> in web.config):
```
<rewrite>
  <rules>
    <rule name="Set HTTPS var" stopProcessing="true">
      <match url=".*" />
      <conditions logicalGrouping="MatchAll" trackAllCaptures="false">
        <add input="{HTTP_X_FORWARDED_PROTO}" pattern="^https$"
negate="true" /> <add input="{HTTP_X_FORWARDED_PROTO}" pattern="https"
ignoreCase="true" /> </conditions>
      <serverVariables>
        <set name="HTTPS" value="on" />
      </serverVariables>
      <action type="None" />
    </rule>
    </rules>
  <allowedServerVariables>
       <add name="HTTPS" />
  </allowedServerVariables>
</rewrite>
```

Explanation:
- This rule checks if the X-Forwarded-Proto header exists and its value is https.
- If true, it sets an HTTPS server variable to on.
- This variable can then be passed to the Python process via the <environmentVariables> section in <httpPlatform> (as shown commented out in the previous web.config example).
- The python3-saml library often checks for this HTTPS environment variable (or similar WSGI environment keys) to determine the correct scheme for URLs.
- You might need additional rules to handle the X-Forwarded-Host header or other proxy-specific headers depending on your exact Azure networking setup (Load Balancer, Application Gateway, Front Door).
- The <allowedServerVariables> section is necessary to permit URL Rewrite rules to modify specific server variables.

Testing: Thoroughly test the SAML flow after deploying to IIS/Azure. Use browser developer tools and SAML tracers (Step 7.2) to inspect URLs and headers at each step. Check the stdoutLogFile for Python startup errors and Django logs for runtime issues.

Step 5: Understanding the Authentication Flow

Visualizing the sequence of events during a SAML login helps in configuration and troubleshooting. Here's a typical **Service Provider (SP)-Initiated** flow using Entra ID and your Django application:
1. **User Accesses Django App:** The user navigates to a protected resource in your Django application or explicitly clicks a "Login with Entra ID" button/link that points to your SAML login initiation URL (e.g., /saml/login/).
2. **SP Initiates SSO:** The corresponding Django view (using python3-saml) constructs a SAML Authentication Request (AuthnRequest). This XML message asks the IdP to

authenticate the user and specifies details like the SP's Entity ID and the ACS URL where the response should be sent.

3. **Redirect to IdP:** Django sends an HTTP Redirect response to the user's browser. The redirect URL is Entra ID's Single Sign-On endpoint (the **Login URL** obtained in Step 2.7), with the AuthnRequest typically encoded and included as a query parameter (SAMLRequest).
4. **User Authenticates at IdP:** The browser follows the redirect to the Entra ID login page. The user enters their corporate credentials (username, password) and completes any required Multi-Factor Authentication (MFA). Entra ID validates these credentials against its directory.
5. **IdP Generates SAML Response:** Upon successful authentication, Entra ID generates a SAML Response. This XML document contains the SAML Assertion, which includes:
    - The user's unique identifier (NameID) in the format requested by the SP.
    - The configured attributes/claims (email, name, group IDs, etc.).
    - Conditions defining the assertion's validity period and intended audience (which must match the SP's Entity ID).
    - Authentication context information (how the user authenticated).
    - A digital signature created using Entra ID's private signing key.
6. **IdP Sends Response to SP:** Entra ID sends the SAML Response back to the user's browser, typically embedded within an HTML form that automatically POSTs to the Django application's Assertion Consumer Service (ACS) URL (the **Reply URL** configured in Entra ID, e.g., /saml/acs/). The RelayState parameter, if originally sent by the SP, is usually included in this POST request to maintain application state.
7. **SP Processes Response:** The Django ACS view (e.g., /saml/acs/) receives the HTTP POST request containing the SAMLResponse parameter. It uses the python3-saml library to perform critical validation steps:
    - Parses the XML Response.
    - **Verifies the Signature:** Uses Entra ID's public certificate (configured in settings.py from Step 2.7) to validate the digital signature on the assertion or response message. This ensures the response genuinely came from Entra ID and wasn't tampered with.
    - **Validates Conditions:** Checks timestamps (NotBefore, NotOnOrAfter) to prevent replay attacks and ensures the assertion hasn't expired.
    - **Validates Audience:** Confirms the AudienceRestriction in the assertion matches the SP's Entity ID.
    - **Validates Destination:** Checks that the Destination attribute in the Response matches the SP's ACS URL where the response was received (this is where proxy issues often manifest).
    - **Extracts Information:** If all checks pass, extracts the NameID and attributes (claims) from the assertion.
8. **User Session Created:** The Django view uses the extracted NameID or a designated unique attribute (e.g., email from SAML_USERNAME_ATTR) to find or create a corresponding user in the Django database via the SamlUserBackend. It maps other extracted attributes (first name, last name) to the user object based on SAML_ATTR_MAP. Finally, it uses django.contrib.auth.login() to establish an authenticated session for the user and redirects the browser to the appropriate page (either the original destination stored in RelayState or the LOGIN_REDIRECT_URL).

The user is now logged into the Django application without having entered their password directly into it.

Step 6: Implementing Authorization in Django

Authentication confirms the user's identity, but authorization determines what actions the authenticated user is permitted to perform within the application. SAML assertions can carry authorization information, most commonly through group membership claims, which can be leveraged by your Django application.

Using Group Claims from Entra ID

In Step 2.5, you configured Entra ID to send a groups claim containing the Object IDs of the groups the user belongs to (specifically, the groups assigned to the application, if configured that way). Using these immutable Object IDs is the recommended approach for authorization based on Entra ID groups.
You can use this information in Django to control access:

1. **Store Group Information:** During the SAML login process (within your ACS view or a custom authentication backend inheriting from SamlUserBackend), after successfully validating the SAML response, extract the list of group Object IDs from the groups attribute in the assertion data (auth.get_attributes()['groups']). Store this list somewhere accessible during the user's session, such as:
    - The Django session (request.session['saml_groups'] = group_ids).
    - A field on a custom user profile model linked to the Django User model.
2. **Check Membership in Views/Middleware:** In your Django views, middleware, or custom permission decorators, retrieve the user's list of Entra ID group IDs and check if a required group ID is present before allowing access to a resource or action.

Example Code (in a Django view):
```
from django.shortcuts import render
from django.contrib.auth.decorators import login_required
from django.core.exceptions import PermissionDenied

# Assume this is the Object ID of the Entra ID group that grants admin
access
# Find this in the Entra ID portal under Groups -> Your Group ->
Overview -> Object Id
ADMIN_GROUP_OBJECT_ID = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"

@login_required # Ensure user is authenticated (via SAML or other
means)
def admin_dashboard(request):
    # Retrieve group IDs stored during SAML login
    # This assumes they were stored in the session. Adjust if using a
profile model.
    user_entra_group_ids = request.session.get('saml_groups',)

    if ADMIN_GROUP_OBJECT_ID not in user_entra_group_ids:
        # If the required group ID is not found, deny access
        raise PermissionDenied("You do not have permission to access 
the admin dashboard.")

    # User is in the required Entra ID group, proceed with the view 
logic
    context = {
        'message': 'Welcome to the Admin Dashboard!'
    }
    return render(request, 'app/admin_dashboard.html', context)

# In your ACS view (where SAML response is processed):
# After auth.is_authenticated() is True:
# saml_attributes = auth.get_attributes()
# entra_group_ids = saml_attributes.get('groups',) # Get the list of 
group IDs
# request.session['saml_groups'] = entra_group_ids # Store in session
#... proceed with user lookup/creation and login...
```

This approach directly uses the group information provided by Entra ID.

Using Other Attributes for Authorization

Similarly, you could configure Entra ID to send a custom claim representing a user's role (e.g., role claim with value administrator or editor) and check this attribute's value in your Django application.

Mapping to Django Groups and Permissions

An alternative, more decoupled approach involves mapping the incoming Entra ID group Object IDs to Django's built-in Group objects during the login process (in the ACS view or custom backend).

1. Define corresponding django.contrib.auth.models.Group objects in your Django application (e.g., an "Entra Admins" Django group).
2. In the ACS view, after extracting the Entra ID group IDs:
    - Check if the user's Entra ID group list contains the ADMIN_GROUP_OBJECT_ID.
    - If yes, fetch or create the corresponding Django "Entra Admins" group and add the Django user to it (user.groups.add(django_admin_group)).
    - Remove the user from groups they are no longer part of according to the SAML assertion.
3. Assign permissions to the Django "Entra Admins" group using Django's standard admin interface or migrations.
4. Use standard Django permission checks in your views (e.g., @permission_required('app.can_view_dashboard'), user.has_perm('app.can_view_dashboard')).

This method leverages Django's authorization system more directly but requires implementing and maintaining the mapping logic during login.

Consideration: Entra ID Group Limit

Remember the Entra ID limit on the number of groups sent in the token. If a user exceeds this limit, the groups claim might be incomplete. Relying solely on the claim for critical authorization could be risky in such cases. Strategies to mitigate this include:

- Using "Groups assigned to the application" in the Entra ID claim configuration to limit the number of groups sent.
- Designing authorization around a smaller number of critical role-based groups.
- Implementing fallback logic to query the Microsoft Graph API for full group membership if the claim seems incomplete (this adds complexity).

For a beginner setup, directly checking the received group Object IDs is often the simplest starting point for authorization.

Step 7: Troubleshooting Common Issues

Integrating SAML can involve several moving parts, and issues can arise. A systematic approach to troubleshooting is crucial.

7.1. Checking Entra ID Sign-in Logs

This is often the first place to look when a login fails.

- **Location:** In the Microsoft Entra admin center, navigate to **Identity** > **Monitoring & health** > **Sign-in logs**.
- **Filtering:** Filter the logs by the specific **User** attempting to sign in and the **Application** (your Django SAML app). Set the **Status** to **Failure**.
- **Analysis:** Examine the failed entries. Key information includes:
    - **Error Code:** Entra ID provides specific error codes (e.g., 50058: Session expired, 50121: MFA failed, 70046: Reauth failed, 90025: Retry limit). Search the Microsoft documentation for the specific code.
    - **Failure Reason:** Provides a textual description of the error (e.g., "Invalid username or password," "Application configuration problem," "Conditional Access policy requirement not met").
    - **Conditional Access:** Check the "Conditional Access" tab for the failed sign-in to see if a policy blocked the attempt.
    - **Authentication Details:** Can show which step of the authentication process failed (e.g., credential validation, MFA).
- **Common Issues Seen Here:** Incorrect credentials, MFA problems, user not assigned to the application, Conditional Access blocks, problems with the application's configuration in Entra ID (e.g., invalid Reply URL).

7.2. Using Browser SAML Tracer Tools

These browser extensions capture the SAML messages exchanged between the browser, SP (Django), and IdP (Entra ID) in real-time, allowing detailed inspection.

- **Tools:**
    - **SAML-tracer** (Firefox Add-on)
    - **SAML Message Decoder** (Chrome Extension)
    - **SAML Chrome Panel** (Chrome Extension)
    - **SAML DevTools extension** (Chrome/Edge)
- **Installation:** Install the appropriate extension for your browser from its web store. You

might need to enable it for incognito/private browsing modes if testing there.

- **Usage:**
    1. Open the browser's Developer Tools (usually F12) and select the SAML tab provided by the extension OR click the extension's icon in the toolbar.
    2. Clear any previous logs.
    3. Initiate the SAML login flow from your Django application (e.g., click the SSO login button).
    4. Authenticate at the Entra ID login page when prompted.
    5. Observe the requests appearing in the tracer window. Look for entries related to your Django app URL and the Entra ID login URL (login.microsoftonline.com). Entries marked with a "SAML" icon indicate SAML messages.
- **Inspection:**
    - **AuthnRequest (SP -> IdP):** Select the HTTP request going from your Django app to Entra ID (often a GET request if using HTTP-Redirect). Check the "SAML" tab in the tracer. Verify:
        - Issuer: Matches your SP's Entity ID.
        - Destination: Matches Entra ID's SSO URL.
        - AssertionConsumerServiceURL: Matches your SP's ACS URL.
        - Presence of signature if authnRequestsSigned is true.
    - **SAMLResponse (IdP -> SP):** Select the HTTP POST request going from Entra ID back to your Django app's ACS URL. Check the "SAML" tab. Verify:
        - Issuer: Matches Entra ID's Entity ID.
        - Destination: Matches your SP's ACS URL (critical!).
        - Audience: Matches your SP's Entity ID.
        - Signature: Check if the assertion and/or response is signed as expected.
        - StatusCode: Should indicate success (e.g., urn:oasis:names:tc:SAML:2.0:status:Success).
        - NameID: Check the format and value.
        - AttributeStatement: Verify the expected attributes (email, name, groups) are present with the correct names and values.
- **Export:** Most tracers allow exporting the captured session (often as JSON or HAR file) which can be useful for sharing with support or offline analysis.

7.3. Validating Metadata

Incorrect or out-of-sync metadata is a frequent cause of SAML errors.

- **SP Metadata:**
    - Access the metadata URL provided by your Django app (e.g., /saml/metadata/).
    - Verify the XML content: Does the entityID match Entra ID's Identifier? Do the AssertionConsumerService and SingleLogoutService URLs match the Reply URL and Logout URL in Entra ID, including binding types (HTTP-POST for ACS, HTTP-Redirect for SLS are common with python3-saml)? Is the correct SP signing certificate included if applicable?
- **IdP Metadata:**
    - If using SAML_IDP_URL in Django, ensure the URL is correct, publicly accessible from your Azure VM, and points to the metadata for *your specific application registration* in Entra ID.
    - If using SAML_IDP_METADATA_FILE, ensure it's the correct, up-to-date file

    downloaded from your Entra ID application settings.
  - If using manual SAML_IDP dictionary, double-check all URLs, the Entity ID, and the x509cert string against the values shown in the Entra ID portal for your application.
- **Online Validation Tools:** Use tools like samltool.io , OneLogin's SAML tools , or saml-validator to check the syntax and basic compliance of your SP or IdP metadata XML. You can often paste the XML directly or provide the URL. Some tools can also decode and validate SAML requests/responses. *Exercise caution when pasting sensitive data or production tokens into public online tools.*

7.4. Verifying Certificate Configuration

Certificate issues often lead to signature validation failures.

- **IdP Certificate in SP Config:** The most common issue is having the wrong IdP signing certificate configured in your Django settings.py (SAML_IDP['x509cert'] or fetched via metadata). Ensure you have copied the **correct, active Base64 certificate** from the **SAML Certificates** section of your Entra ID application. Double-check for copy-paste errors, extra whitespace, or missing characters. Ensure the format matches what the library expects (raw Base64 string vs. file path).
- **Certificate Expiration:** Check if the IdP certificate configured in Django or the SP certificate (if used) has expired. Entra ID provides notifications for expiring certificates if configured.
- **Certificate Rollover:** Entra ID rotates certificates periodically. If you are not using the metadata URL (which usually updates automatically), you need to manually update the certificate in your Django settings when Entra ID activates a new one. Some applications might cache metadata/certificates, requiring a refresh or restart. Entra ID sometimes allows configuring rollover behavior.
- **SP Certificate Issues:** If your SP signs requests (authnRequestsSigned=True), ensure the privateKey and x509cert paths in SAML_SP are correct and the files exist with proper permissions. If Entra ID is configured to *require* signed requests, ensure signing is enabled in Django.
- **Trust Issues:** Ensure the certificate chain is valid and trusted if using certificates not directly issued by a well-known CA (less common with Entra ID's default certs, but relevant for custom certs or self-signed certs in testing).
- **Clock Skew:** Significant time differences (> few minutes) between the Azure VM hosting Django and Entra ID servers can cause assertion validation failures based on NotBefore and NotOnOrAfter timestamps. Ensure the VM's clock is synchronized using NTP.

7.5. Common Configuration Pitfalls

- **URL Mismatches:** Slight differences (HTTP vs. HTTPS, trailing slash, incorrect path) between the Identifier (Entity ID), Reply URL (ACS), Sign on URL, or Logout URL in Entra ID versus your Django settings.py (SAML_SP) and urls.py. Verify exact matches.
- **Claim/Attribute Name Mismatches:** Django code expects an attribute named email, but Entra ID is sending http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress. Update SAML_ATTR_MAP in settings.py to use the exact name sent by Entra ID. Use a SAML tracer to confirm attribute names.
- **User Not Assigned:** Forgetting to assign the test user account (or their group) to the Enterprise Application in Entra ID (if "Assignment required?" is set to Yes).

- **IIS Configuration Errors:** Problems in web.config (incorrect Python path, wrong arguments for WSGI server, missing %HTTP_PLATFORM_PORT%), permissions issues for the log directory or project files, failure of the WSGI server to start. Check the stdoutLogFile defined in web.config for errors.
- **Proxy/Header Issues:** If behind a load balancer/proxy, failure to configure IIS URL Rewrite (or equivalent) to handle X-Forwarded-Proto/X-Forwarded-Host can lead to Destination validation errors or incorrect metadata generation.
- **Library Dependencies:** Failure to install xmlsec1 correctly on the server.

Conclusion

This report has detailed the process of integrating a Django application, hosted on IIS within an Azure VM, with Microsoft Entra ID using the SAML 2.0 protocol for Single Sign-On. By following the steps outlined – understanding SAML fundamentals, registering and configuring the Entra ID enterprise application, selecting and configuring a Python SAML library (python3-saml via python3-saml-django) within Django, setting up IIS with HttpPlatformHandler and considering proxy implications, understanding the authentication flow, implementing basic group-based authorization, and utilizing troubleshooting techniques – developers can successfully leverage Entra ID for robust and user-friendly authentication.
Key takeaways include the critical importance of exact URL matching between Entra ID and Django configurations, the correct mapping of claims/attributes, the necessity of using a production WSGI server with HttpPlatformHandler, and the potential need for IIS URL Rewrite when operating behind proxies. Careful attention to certificate management and utilizing tools like Entra ID logs and SAML tracers are essential for debugging.
While this guide covers the core integration, further steps could involve implementing robust Single Logout (SLO), automating certificate rotation management, exploring advanced claim transformations within Entra ID, or deeper integration with Django's permission framework by mapping Entra ID groups to Django groups.
Successfully implementing SAML SSO enhances application security by centralizing authentication, improves user experience by reducing password fatigue, and simplifies user management for administrators.

Works cited

1. What is SAML? | How SAML authentication works - Cloudflare, https://www.cloudflare.com/learning/access-management/what-is-saml/ 2. What is SAML 2.0 and how does it work for you? - Auth0, https://auth0.com/intro-to-iam/what-is-saml 3. SAML Explained in Plain English - OneLogin, https://www.onelogin.com/learn/saml 4. SAML authentication with Microsoft Entra ID, https://learn.microsoft.com/en-us/entra/architecture/auth-saml 5. Introduction to SAML v2.0 | AM 7.2.2 - Ping Identity Docs, https://docs.pingidentity.com/pingam/7.2/saml2-guide/saml2-introduction.html 6. What is SAML and how does SAML Authentication Work | Auth0, https://auth0.com/blog/how-saml-authentication-works/ 7. SAML 2 0 Technical Overview - YouTube, https://www.youtube.com/watch?v=e61yFzxcL8A 8. Single sign-on SAML protocol - Microsoft identity platform, https://learn.microsoft.com/en-us/entra/identity-platform/single-sign-on-saml-protocol 9.

Microsoft Entra Connect: Use a SAML 2.0 Identity Provider for Single Sign On - Azure, https://learn.microsoft.com/en-us/entra/identity/hybrid/connect/how-to-connect-fed-saml-idp 10. Microsoft Entra ID (Using SAML) – Posit Connect Documentation ..., https://docs.posit.co/connect/admin/authentication/saml-based/entra-id-saml/ 11. How the Microsoft identity platform uses the SAML protocol, https://learn.microsoft.com/en-us/entra/identity-platform/saml-protocol-reference 12. Microsoft Entra federation metadata - Microsoft identity platform, https://learn.microsoft.com/en-us/entra/identity-platform/federation-metadata 13. Download Metadata URL from Microsoft Entra ID - Laserfiche, https://doc.laserfiche.com/laserfiche/en-us/Content/DownloadMetadataURLfromAzureAD.htm 14. SAML-Toolkits/python-saml - GitHub, https://github.com/SAML-Toolkits/python-saml 15. OneLogin SAML Python Toolkit 2.0.0 documentation - Pythonhosted.org, https://pythonhosted.org/python-saml/ 16. Register a SAML app in your external tenant (preview) - Learn Microsoft, https://learn.microsoft.com/en-us/entra/external-id/customers/how-to-register-saml-app 17. Enable SAML single sign-on for an enterprise application - Microsoft Entra ID, https://learn.microsoft.com/en-us/entra/identity/enterprise-apps/add-application-portal-setup-sso 18. Tutorial: How to Configure SAML with Entra ID - OpenVPN, https://openvpn.net/as-docs/tutorials/saml-entra-id.html 19. How to Set Up SAML Single Sign-On with Microsoft Entra ID, https://knowledgebase.businessmap.io/hc/en-us/articles/115004167265-How-to-Set-Up-SAML-Single-Sign-%D0%9En-with-Microsoft-Entra-ID 20. Example: SAML With Microsoft Entra ID - ProcessMaker, https://docs.processmaker.com/docs/example-saml-with-microsoft-entra-id 21. Security Assertion Markup Language (SAML) single sign-on (SSO) for on-premises apps with Microsoft Entra application proxy, https://learn.microsoft.com/en-us/entra/identity/app-proxy/conceptual-sso-apps 22. Entra ID Applications: Basic Configuration - UW-IT, https://uwconnect.uw.edu/it?id=kb_article_view&sysparm_article=KB0034047 23. Set up Single Sign-On (SSO) for Entra ID (formerly Azure Active Directory) - Skillable, https://docs.skillable.com/docs/set-up-single-sign-on-sso-for-entra-id-formerly-azure-active-directory-1 24. IronCountySchoolDistrict/django-python3-saml - GitHub, https://github.com/IronCountySchoolDistrict/django-python3-saml 25. penn-state-dance-marathon/python3-saml-django - GitHub, https://github.com/penn-state-dance-marathon/python3-saml-django 26. SSO/SAML Authentication with Microsoft Entra ID (formerly Azure Active Directory) - Gridly, https://help.gridly.com/4417665404049-sso-saml-authentication-with-microsoft-entra-id 27. Configuring SAML SSO with Microsoft Entra ID - Cisco Meraki Documentation, https://documentation.meraki.com/General_Administration/Managing_Dashboard_Access/Configuring_SAML_SSO_with_Microsoft_Entra_ID 28. Entra ID SAML (formerly Azure AD) – Integrations – WorkOS Docs, https://workos.com/docs/integrations/entra-id-saml 29. Example: SAML Based Authentication with Microsoft Entra ID, https://docs.axonius.com/docs/example-saml-based-authentication-with-azure-active-directory 30. Configure Single Sign-On (SAML) with Microsoft Entra ID and Microsoft 365, https://community.forumbee.com/t/35hy6db/configure-single-sign-on-saml-with-azure-active-directory-and-microsoft-365 31. Unable to set Identifier (Entity ID) or Reply URL for Azure SAML SSO configuration, https://learn.microsoft.com/en-us/answers/questions/450746/unable-to-set-identifier-(entity-id)-or-reply-url 32. Troubleshoot SAML-based single sign-on - Microsoft Entra ID, https://learn.microsoft.com/en-us/entra/identity/enterprise-apps/troubleshoot-saml-based-sso 33. Configuring Microsoft Entra ID as a SAML Identity Provider - TechDocs - Broadcom Inc., https://techdocs.broadcom.com/us/en/vmware-tanzu/platform-services/single-sign-on-for-tanzu/1-16/sso-tanzu/azure-config-azure.html 34. How to configure Microsoft Entra ID SAML SSO - Calendly Help, https://help.calendly.com/hc/en-us/articles/1500001345241-How-to-configure-Microsoft-Entra-ID-SAML-SSO 35. Entra ID - Configuring Single Sign-On Authentication | Staffbase Developer Portal, https://developers.staffbase.com/guides/configuring-sso/ 36. Example of user configuration with SAML 2.0 and Entra ID - Bizzdesign Support, https://help.bizzdesign.com/articles/knowledge-base/example-of-user-configuration-with-saml-2-0-and-entra-id 37. Customize SAML token claims - Microsoft identity platform | Microsoft ..., https://learn.microsoft.com/en-us/entra/identity-platform/saml-claims-customization 38. Set up your own custom SAML application for Microsoft Entra ID SAML - BoxyHQ, https://boxyhq.com/docs/jackson/sso-providers/azure 39. Configuring Microsoft Entra ID for Deep Discovery Inspector (DDI) SAML Authentication, https://success.trendmicro.com/en-US/solution/ka-0014995 40. Configuring SAML for Microsoft Entra ID - Zscaler Help Portal, https://help.zscaler.com/itdr/configuring-saml-microsoft-entra-id 41. Configure group claims for applications by using Microsoft Entra ID, https://learn.microsoft.com/en-us/entra/identity/hybrid/connect/how-to-connect-fed-group-claims 42. Configuring Microsoft Entra ID as a SAML identity provider | Qlik Cloud Help, https://help.qlik.com/en-US/cloud-services/Subsystems/Hub/Content/Sense_Hub/Admin/configuring-idp-entraid.htm 43. Microsoft Entra ID - Dizzion Documentation, https://docs.dizzion.com/platform/identity-and-access/idp-integrations/entra-id 44. Configure Single Sign-on (SSO) with Microsoft Entra ID, https://stackoverflowteams.help/en/articles/4538487-configure-single-sign-on-sso-with-microsoft-entra-id 45. SAML SSO with Microsoft Entra ID (Azure Active Directory) - Infosec IQ, https://support.infosecinstitute.com/s/article/SAML-SSO-with-Azure-Active-Directory 46. Tutorial: Manage federation certificates - Microsoft Entra ID, https://learn.microsoft.com/en-us/entra/identity/enterprise-apps/tutorial-manage-certificates-for-federated-single-sign-on 47. Create an Enterprise Application in Microsoft Entra ID, https://support.ptc.com/help/identity_and_access_management/en/iam/AzureADCreateEnterpriseApp_Windchill.html 48. How to Set Up an Microsoft Entra ID to SAML Integration - Panopto Support, https://support.panopto.com/s/article/How-to-Set-Up-an-Azure-AD-to-SAML-Integration 49. Configure SAML single sign-on with Microsoft Entra ID - Deep Security Help Center, https://help.deepsecurity.trendmicro.com/20_0/on-premise/saml-sso-azure-ad.html 50. How to retrieve your Azure AD SAML 2.0 metadata - Customer Support Portal, https://support.montycloud.com/support/solutions/articles/62000206239-how-to-retrieve-your-azure-ad-saml-2-0-metadata 51. Python SSO: pysaml2 and python3-saml - Stack Overflow, https://stackoverflow.com/questions/40617347/python-sso-pysaml2-and-python3-saml 52. ITS Operations : SSO - OneLogin Python SAML Toolkit, https://shib02a.rit.edu/ITSOperations/SSO---OneLogin-Python-SAML-Toolkit_22252902.html 53. SAML-Toolkits/python3-saml - GitHub, https://github.com/SAML-Toolkits/python3-saml 54. IdentityPython/pysaml2: Python implementation of SAML2 - GitHub, https://github.com/IdentityPython/pysaml2 55. python3-saml-django · PyPI, https://pypi.org/project/python3-saml-django/ 56. python3-saml-django/setup.py at master - GitHub, https://github.com/penn-state-dance-marathon/python3-saml-django/blob/master/setup.py 57. SAML SSO for Django - Curious DBA, https://curiousdba.netlify.app/post/djangosaml/ 58.

SAML2 & Python 3.11+, do I have to use a third party library? - Okta Developer Community, https://devforum.okta.com/t/saml2-python-3-11-do-i-have-to-use-a-third-party-library/26914 59. I am trying to install python-saml. It is throwing build dependencies error. It does not give specific reason why it is failing, https://stackoverflow.com/questions/77483413/i-am-trying-to-install-python-saml-it-is-throwing-build-dependencies-error-it 60. IdentityPython/djangosaml2: Django SAML2 Service Provider based on pySAML2 - GitHub, https://github.com/IdentityPython/djangosaml2 61. djangosaml2 - PyPI, https://pypi.org/project/djangosaml2/ 62. FreshPorts -- www/py-djangosaml2: Pysaml2 integration for Django, https://www.freshports.org/www/py-djangosaml2/ 63. Welcome to Djangosaml2's Documentation — djangosaml2 0.2.4 documentation, https://djangosaml2.readthedocs.io/ 64. Setup — djangosaml2 0.2.4 documentation, https://djangosaml2.readthedocs.io/contents/setup.html 65. djangosaml2-bernii · PyPI, https://pypi.org/project/djangosaml2-bernii/ 66. Implementing Single Sign-On (SSO) with SAML for a Django Application, https://wersdoerfer.de/blogs/ephes_blog/implementing-single-sign-on-sso-with-saml-for-a-django-application/ 67. wfastcgi - PyPI, https://pypi.org/project/wfastcgi/ 68. Running a Django Application on Windows Server 2012 with IIS - Matt Woodward, https://www.mattwoodward.com/2016/07/23/running-a-django-application-on-windows-server-2012-with-iis/ 69. Getting 500 Internal Server Error when setting up Python and Flask with FastCgiModule on Windows - Stack Overflow, https://stackoverflow.com/questions/36744531/getting-500-internal-server-error-when-setting-up-python-and-flask-with-fastcgim 70. Django IIS deployment with HttpPlatformHandler, https://forum.djangoproject.com/t/django-iis-deployment-with-httpplatformhandler/26824 71. HttpPlatformHandler error with Django app hosted in IIS - Stack Overflow, https://stackoverflow.com/questions/77752026/httpplatformhandler-error-with-django-app-hosted-in-iis 72. Running Django Web Apps on IIS with HttpPlatformHandler - LeXtudio Documentation, https://docs.lextudio.com/blog/running-django-web-apps-on-iis-with-httpplatformhandler/ 73. HttpPlatformHandler Configuration Reference | Microsoft Learn, https://learn.microsoft.com/en-us/iis/extensions/httpplatformhandler/httpplatformhandler-configuration-reference 74. iis - URL Rewrite Not Firing Properly - Server Fault, https://serverfault.com/questions/881107/url-rewrite-not-firing-properly 75. python django Mock SAML Response from onelogin.saml.auth library using python3-saml, https://stackoverflow.com/questions/63724493/python-django-mock-saml-response-from-onelogin-saml-auth-library-using-python3-s 76. Setting HTTP request headers and IIS server variables | Microsoft Learn, https://learn.microsoft.com/en-us/iis/extensions/url-rewrite-module/setting-http-request-headers-and-iis-server-variables 77. SP-Initiated SAML SSO Configuration Guide - Cisco Meraki Documentation, https://documentation.meraki.com/General_Administration/Managing_Dashboard_Access/SP-Initiated_SAML_SSO_Configuration_Guide 78. Set up SAML SSO with Microsoft Entra ID (formerly Azure AD) - Apollo GraphQL Docs, https://www.apollographql.com/docs/graphos/platform/access-management/sso/saml-microsoft-entra-id 79. How to debug an Invalid Signature on SAML Response - Stack Overflow, https://stackoverflow.com/questions/48797528/how-to-debug-an-invalid-signature-on-saml-response 80. Failed authentication with SAML Certificate - Microsoft Community Hub, https://techcommunity.microsoft.com/discussions/microsoft-entra/failed-authentication-with-saml-certificate/4291502 81. SAML Login Errors - Oracle Help Center, https://docs.oracle.com/en-us/iaas/Content/Identity/troubleshooting/saml_login_errors/saml-login-errors.htm 82. django-saml-sp - PyPI, https://pypi.org/project/django-saml-sp/ 83. grafana/django-saml2-auth: Django SAML2 Authentication Made Easy. Easily integrate with SAML2 SSO identity providers like Okta, Azure AD and others. - GitHub, https://github.com/grafana/django-saml2-auth 84. Example app with SAML support, built with Python + Django + SSOReady - GitHub, https://github.com/ssoready/ssoready-example-app-python-django-saml 85. How to Troubleshoot Common Microsoft Entra ID Issues - AdminDroid Blog, https://blog.admindroid.com/12-common-microsoft-entra-id-issues-fixes-for-admins/ 86. Common Issues with SAML Authentication - Blackboard Help, https://help.blackboard.com/Learn/Administrator/SaaS/Authentication/Implement_Authentication/SAML_Authentication_Provider_Type/Common_Issues_with_SAML_Authentication 87. How to perform a SAML Trace - Adobe Help Center, https://helpx.adobe.com/enterprise/kb/perform-a-saml-trace.html 88. How to Troubleshoot with SAML Tracer - Okta Support, https://support.okta.com/help/s/article/How-to-troubleshoot-with-SAML-Tracer?language=en_US 89. Test SAML app implementation with SAML-tracer - Okta Developer, https://developer.okta.com/docs/guides/saml-tracer/main/ 90. knowledge.autodesk.com, https://knowledge.autodesk.com/guidref/SSOGUIDE_Okta_Guide_About_Single_Sign_on_SSO_Troubleshooting_how_to_run_a_saml_trace_html#:~:text=Go%20to%20Google%20chrome%20click,details%20of%20the%20selected%20entry. 91. Troubleshooting SAML authentication issues - Posit Support, https://support.posit.co/hc/en-us/articles/6436016954647-Troubleshooting-SAML-authentication-issues 92. samltool.io: SAML Tokens, https://samltool.io/ 93. SAML AuthN Request - OneLogin Developers, https://developers.onelogin.com/saml/online-tools/validate/saml-authnrequest 94. SAML-validator - safeID, https://mdr.safeid.sk/saml-validator/ 95. SAML certificates explained - Stytch, https://stytch.com/blog/saml-certificates/