Beginner's Guide to the TeamViewer AD Check Python Code

Overview

This Python project is a **Flask-based web application** that interacts with Active Directory (AD) to check whether users exist in AD and retrieve key details (name, office, department, and title). The results are processed in real-time and exported as an Excel file.

This guide explains the code step-by-step for **beginner programmers** who want to understand how it works.

---

1. Setting Up the Environment

Before running the code, you need to install the required Python libraries. Run the following command:

```
pip install flask ldap3 pandas openpyxl
```

These libraries provide:

- **Flask** – A web framework to create the user interface.
- **ldap3** – To interact with Active Directory (AD).
- **pandas** – To handle tabular data and create an Excel file.
- **openpyxl** – To export results to Excel.

---

2. Understanding the Main Components

📌 Flask App Initialization

At the beginning of the script, we import the necessary libraries and create the Flask app:

```
from flask import Flask, render_template, request, send_file, Response, stream_with_context

import pandas as pd

import ldap3

import os

import socket

import getpass

from io import BytesIO

import time


app = Flask(__name__)
```

✅ **Flask** will handle web requests and responses.

✅ **pandas** and **BytesIO** will be used to generate the Excel file.

📌 Detecting LDAP Server & Search Base

To interact with AD, we need to know the **LDAP server** and the **search base (domain structure)**. This function detects them automatically:

```
def get_ldap_details():
  try:
    result = os.popen("nslookup -type=SRV _ldap._tcp").read()
    lines = result.split("\n")
    ldap_server = ""
    search_base = ""

    for line in lines:
      if "svr hostname" in line.lower():
        ldap_server = line.split()[-1].strip()
        break

    if ldap_server and "." in ldap_server:
      domain_parts = ldap_server.split(".")[1:]  # Ignore DC hostname
      search_base = ",".join([f"DC={part}" for part in domain_parts])

    return ldap_server, search_base
  except Exception as e:
    print(f"Error detecting LDAP server: {e}")
  return "", ""
```

✅ Uses nslookup to find the **LDAP server**.

✅ Extracts the **domain components** to form the correct **LDAP search base** (e.g., DC=company,DC=local).

📌 **Detecting the Logged-in User**

We want to **auto-populate** the username in the format DOMAIN\username:

```
def get_current_user():
  try:
    domain = os.environ.get('USERDOMAIN', '')
    username = getpass.getuser()
    return f"{domain}\\{username}" if domain else username
  except Exception as e:
    print(f"Error getting current user: {e}")
    return getpass.getuser()
```

✅ Fetches the **Windows username** and **domain name**.

✅ Formats it as DOMAIN\username.

📌 **Searching Active Directory**

This function connects to AD and searches for users based on email:

```
def search_ad(email, ldap_server, ldap_user, ldap_password, ldap_search_base):
  server = ldap3.Server(ldap_server)
  conn = ldap3.Connection(
    server,
    user=ldap_user,
    password=ldap_password,
    authentication=ldap3.NTLM,
    auto_bind=True
  )
  search_filter = f'(mail={email})'
  conn.search(ldap_search_base, search_filter, attributes=['displayName',
'physicalDeliveryOfficeName', 'department', 'title'])

  if conn.entries:
    entry = conn.entries[0]
    return {
      'Name': entry.displayName.value if entry.displayName else '',
      'Office': entry.physicalDeliveryOfficeName.value if entry.physicalDeliveryOfficeName else '',
      'Department': entry.department.value if entry.department else '',
      'Title': entry.title.value if entry.title else ''
    }
  return None
```

✅ **Binds to AD** using NTLM authentication.

✅ **Searches for users by email** and retrieves key properties.

📌 **Streaming Progress Updates**

Instead of waiting until all users are processed, this function **streams real-time updates** to the webpage:

```
def generate_progress(emails, ldap_server, ldap_user, ldap_password, ldap_search_base):
  results = []
  total = len(emails)

  for index, email in enumerate(emails, start=1):
    user_info = search_ad(email, ldap_server, ldap_user, ldap_password, ldap_search_base)
    results.append({
      'Email': email,
      'Name': user_info.get('Name', 'Not Found') if user_info else 'Not Found',
      'Office': user_info.get('Office', '') if user_info else '',
      'Department': user_info.get('Department', '') if user_info else '',
      'Title': user_info.get('Title', '') if user_info else ''
    })
    yield f"data: Processing {index}/{total}\n\n"
    time.sleep(0.1)
```

✅ **Processes emails one by one** instead of waiting for all to complete.

✅ **Streams status updates** so the webpage shows real-time progress.

📌 Flask Routes (Web Pages)

Flask handles different web pages and actions:

```
@app.route('/')
def index():
   detected_ldap_server, detected_search_base = get_ldap_details()
   current_user = get_current_user()
   return render_template('index.html', detected_ldap_server=detected_ldap_server, detected_search_base=detected_search_base, current_user=current_user)
```

✅ Loads the **main web page** and pre-fills form fields.

📌 Running the Flask App

Finally, the script runs the Flask web server:

```
if __name__ == '__main__':
   app.run(debug=True, port=5500)
```

✅ Starts the web app at **http://127.0.0.1:5500**.

Final Notes for Beginners

- The app **auto-detects LDAP settings** to reduce manual input.
- Uses **real-time updates** to improve user experience.
- Generates a **downloadable Excel file** with results.
- You can expand this by **adding more AD attributes** or **customizing the UI**.