3. **Describe how to prevent Server-Side Request Forgery (SSRF) attacks in a web application. Write a Python function that demonstrates how to avoid SSRF attacks in a web application.**

PREREQUISITES:

- Python installed on the system.
- Flask library installed.

SETUP:

- Copy the code into a Python file.
- Install the Flask library using the command "pip install Flask".
- Run the Python file to start the Flask application.

ANSWER:

To prevent Server-Side Request Forgery (SSRF) attacks in a web application, you can take the following measures:

1. Validate and sanitize user input: Always validate and sanitize user input before using it to make requests to external resources. Use strict whitelists for URLs, and reject any request that does not meet the requirements.
2. Use a firewall: Configure your web application's firewall to block requests to local IP addresses, private networks, and loopback addresses.
3. Use a reverse proxy server: Set up a reverse proxy server to intercept and validate incoming requests. The proxy server can be configured to block all requests to non-whitelisted resources.
4. Use strict content-type validation: Validate the content-type of the request to ensure that it is a valid URL before making a request. For instance, if the request is expecting an image, the content-type should be an image, and not a URL.
5. Use random, unique, and unpredictable URLs: Use unique and unpredictable URLs instead of predictable ones to prevent attackers from guessing the location of sensitive resources.
6. Set up access controls: Limit access to sensitive resources to specific users, groups, or IP addresses.
7. Monitor and log all requests: Monitor incoming requests and log all requests that are made to external resources. This will help you identify any malicious activity and take appropriate action.

By following these best practices, you can prevent SSRF attacks in your web application and ensure that your sensitive resources are protected.

CODE:

```
from flask import Flask, request
import re

app = Flask(__name__)

# This function is used to render the main page of the website
@app.route("/")
def index():
    # Get the URL that was entered by the user
    url = request.args.get("url", "")

    # Create the HTML response that will be sent back to the user
    html_response = f"""
        <html>
            <head>
                <title>SSRF Example</title>
            </head>
            <body>
                <h1>Fetching data from: {escape_html(url)}</h1>
                {fetch_data(url) if is_valid_url(url) else "Invalid URL"}
                <form action="/" method="GET">
                    <label for="url">Enter a URL:</label>
                    <input type="text" id="url" name="url" value="{escape_html(url)}">
                    <input type="submit" value="Submit">
                </form>
            </body>
        </html>
    """

    # Return the HTML response to the user
    return html_response

# This function is used to escape special characters in the given string
def escape_html(s):
    """
    Escapes special characters in the given string so that they can be safely included in HTML.
    """
    s = s.replace("&", "&amp;")
    s = s.replace("<", "&lt;")
    s = s.replace(">", "&gt;")
    s = s.replace('"', "&quot;")
    s = s.replace("'", "&#39;")
    return s
```

```
# This function is used to check if the given URL is valid
def is_valid_url(url):
   """
   Returns True if the given URL is valid and False otherwise.
   """
   # Use a regular expression to check if the URL starts with either 'http://' or 'https://' and
contains a domain name
   url_regex = re.compile(r"^https?://[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$")
   if url_regex.match(url) is not None:
      return True
   else:
      return False

# This function is used to fetch data from the given URL
def fetch_data(url):
   """
   Fetches data from the given URL and returns it as a string.
   """
   # Add code to fetch data from the URL here
   # ...
   return "Data fetched from URL"

# Start the Flask application
if __name__ == "__main__":
   app.run()
```

EXPLANATION:

This is a Python script that uses the Flask web framework to create a web application that fetches data from a URL entered by the user. The script also includes functions to check if the entered URL is valid and to escape special characters in strings to prevent XSS attacks.

The main functionality of the script is contained in the `index()` function, which is the route for the main page of the website. The function first gets the URL entered by the user from the query string using the `request.args.get()` method. It then creates an HTML response that includes the entered URL and the result of the `fetch_data()` function if the entered URL is valid, and a message indicating that the URL is invalid otherwise.

The `escape_html()` function is used to replace special characters in strings with their corresponding HTML entities, which helps prevent XSS attacks. The `is_valid_url()` function uses regular expressions to check if the entered URL is valid by ensuring that it starts with either "http://" or "https://" and contains a valid domain name.

The `fetch_data()` function is currently a placeholder function that simply returns the string "Data fetched from URL". This function should be replaced with code that actually fetches data from the given URL.

Finally, the script starts the Flask application using the `app.run()` method. When the script is run, the web application will be started and can be accessed in a web browser at the address `http://localhost:5000/`.

PROGRAM OUTCOME:

- The program creates a basic Flask application that allows users to enter a URL and fetch data from it.
- It includes functions to escape special characters in the user input and to check if the URL is valid using regular expressions.
- The program demonstrates how to handle user input and make HTTP requests using the Flask library.