Debugging Python Requests

Jack Henschel

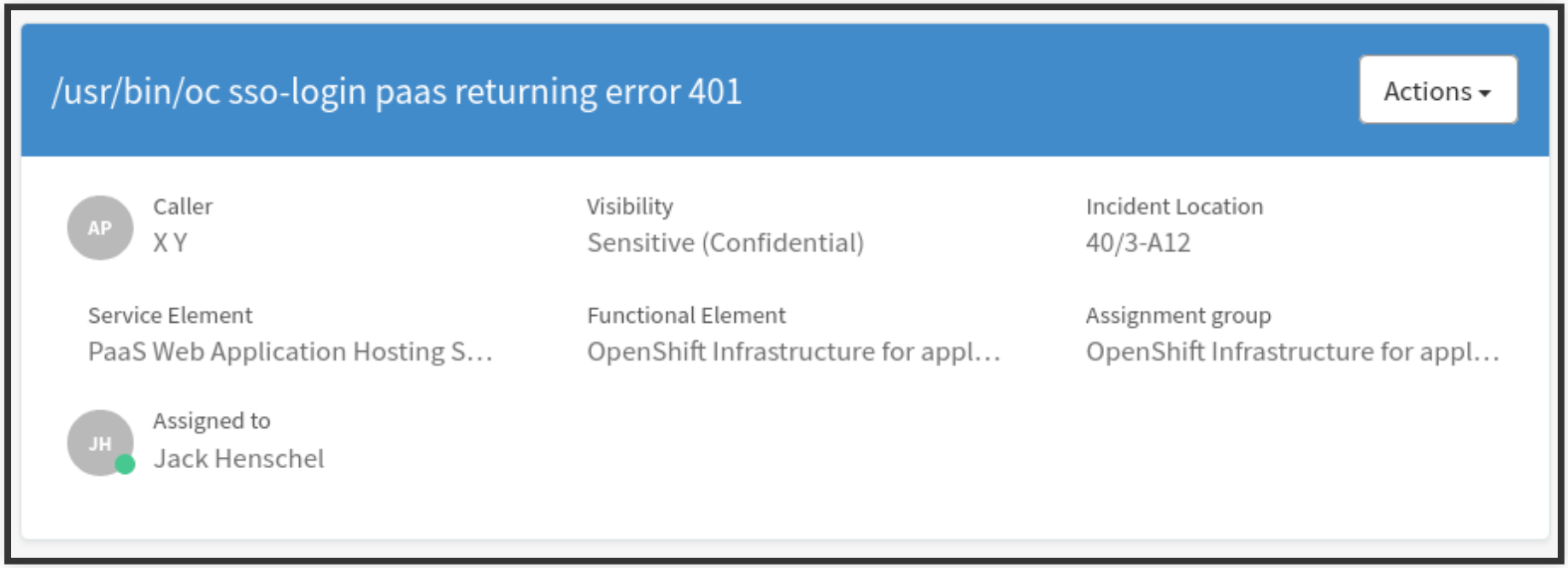
```
$ oc login https://api.paas.okd.cern.ch -u jhensche
Authentication required for https://api.paas.okd.cern.ch:443 (openshift)
Username: jhensche
Password:
Login failed (401 Unauthorized)
Verify you have provided correct credentials.
```

```
$ oc sso-login --server=https://api.paas.okd.cern.ch
Open the following link to log in:

  https://auth.cern.ch/auth/realms/cern/device?user_code=EQGD-UTGC

Waiting for login...
Logged into "https://api.paas.okd.cern.ch:443" as "jhensche" using the token pr

Successfully logged in to 'paas'
```


/usr/bin/oc sso-login paas returning error 401
Actions
AP
Caller
X Y
Visibility
Sensitive (Confidential)
Incident Location
40/3-A12
Service Element
PaaS Web Application Hosting S…
Functional Element
OpenShift Infrastructure for appl…
Assignment group
OpenShift Infrastructure for appl…
JH
Assigned to
Jack Henschel

Jack Henschel

3mo ago • Additional comments (Customer View)

Hello,
Could you clear the token cache and run the command again with the "-v" (verbose) option?
> rm -rf ~/.config/oc-sso_login/*.json
> /usr/bin/oc sso-login paas -v

Please paste the output here (or in attachment) and remove the "access_token" and "refresh_token" values.

```python
def exchange_openshift_token(access_token: str, token_exchange_url: str) -> str:
    params: dict = {'redirect-uri': 'http://localhost'} # dummy value, required by the API
    url: str = f"{token_exchange_url}/openshift-api-token"
    response = requests.get(url, headers={'Authorization': f"Bearer {access_token}"}, params=params)
    if not response.ok:
        print_stderr(response.headers)
        print_stderr(response.text)
        raise Exception(f"Endpoint '{response.url}' returned unexpected status code '{response.status_code}'")

    data: dict = response.json()
    return data['token']

```

Main Interface

All of Requests' functionality can be accessed by these 7 methods. They all return an instance of the **Response** object.

`requests.`**request**(*method, url, **kwargs*) [source]

Constructs and sends a **Request**.

Parameters:

- **method** – method for the new **Request** object: `GET`, `OPTIONS`, `HEAD`, `POST`, `PUT`, `PATCH`, or `DELETE`.
- **url** – URL for the new **Request** object.
- **params** – (optional) Dictionary, list of tuples or bytes to send in the query string for the **Request**.
- **data** – (optional) Dictionary, list of tuples, bytes, or file-like object to send in the body of the **Request**.
- **json** – (optional) A JSON serializable Python object to send in the body of the **Request**.
- **headers** – (optional) Dictionary of HTTP Headers to send with the **Request**.
- **cookies** – (optional) Dict or CookieJar object to send with the **Request**.
- **files** – (optional) Dictionary of `'name': file-like-objects` (or `{'name': file-tuple}`) for multipart encoding upload. `file-tuple` can be a 2-tuple `('filename', fileobj)`, 3-tuple `('filename', fileobj, 'content_type')` or a 4-tuple `('filename', fileobj,`

Hello,

could you check that the content in ${HOME}/.config/oc-sso_login/paas_id_token.json looks reasonable (should be JSON) and in particular that the "access_token" field has a sensible value?
> jq ~/.config/oc-sso_login/paas_id_token.json

If this is the case, I need to understand why the access token is not properly set when making the request to the Openshift API.
To do this, please run the following commands to make a copy of the tool and modify the debug output:

> cp /usr/bin/oc-sso_login ./oc-sso-login-debug
> sed -i 's/print_stderr(response.headers)/print_stderr("headers:", headers, " params:", params)/' oc-sso-login-debug
> python3 ./oc-sso-login-debug paas

Please paste the output here.
Just before the traceback, there should be a line like this:
> headers: {'Authorization': 'Bearer xxx'}

```
84
85  func openshiftApiToken(ctx *gin.Context) {
86          fmt.Println("Headers:", ctx.Request.Header)
87          authorizationHeader := ctx.Request.Header.Get("Authorization")
88          if authorizationHeader == "" {
89                  ctx.JSON(401, gin.H{"error": "authorization header is missing or its value is an empty string"})
90                  return
91          }
92
93          token := strings.Split(authorizationHeader, "Bearer ")
94          if len(token) != 2 {
95                  ctx.JSON(401, gin.H{"error": "value of the Authorization header is invalid"})
96                  return
97          }
98
```

Your client is sending a "Basic" header instead of "Authorization".
This might be a default behavior of python-requests.
Have you configured any .netrc (or similar) on your system?
The Basic authentication header sent by your client is referring to "anonymous:<your-email-address>".

For reference, the headers sent by your client:
Accept:[*/*]
Accept-Encoding:[gzip, deflate]
Authorization:[Basic xxx]
Forwarded:[for="[2001:1458:d00:19::100:145]";host=token-exchange.paas-stg.cern.ch;proto=https]
User-Agent:[python-requests/2.25.1]

and the ones sent from mine (also on LXPLUS9):
Accept:[*/*]
Accept-Encoding:[gzip, deflate]
Authorization:[Bearer xxx]
Forwarded:[for="[2001:1458:d00:19::100:145]";host=token-exchange.paas-stg.cern.ch;proto=https]
User-Agent:[python-requests/2.25.1]

Authentication

This document discusses using various kinds of authentication with Requests.

Many web services require authentication, and there are many different types. Below, we outline various forms of authentication available in Requests, from the simple to the complex.

Basic Authentication

Many web services that require authentication accept HTTP Basic Auth. This is the simplest kind, and Requests supports it straight out of the box.

Making requests with HTTP Basic Auth is very simple:

```
>>> from requests.auth import HTTPBasicAuth
>>> basic = HTTPBasicAuth('user', 'pass')
>>> requests.get('https://httpbin.org/basic-auth/user/pass', auth=basic)
<Response [200]>
```

In fact, HTTP Basic Auth is so common that Requests provides a handy shorthand for using it:

```
>>> requests.get('https://httpbin.org/basic-auth/user/pass', auth=('user', 'pa
<Response [200]>
```

Providing the credentials in a tuple like this is exactly the same as the `HTTPBasicAuth` example above.

netrc Authentication

If no authentication method is given with the `auth` argument, Requests will attempt to get the authentication credentials for the URL's hostname from the user's netrc file. The netrc file overrides raw HTTP authentication headers set with *headers=*.

If credentials for the hostname are found, the request is sent with HTTP Basic Auth.

Digest Authentication

Another very popular form of HTTP Authentication is Digest Authentication, and Requests supports this out of the box as well:

```
>>> from requests.auth import HTTPDigestAuth
>>> url = 'https://httpbin.org/digest-auth/auth/user/pass'
>>> requests.get(url, auth=HTTPDigestAuth('user', 'pass'))
<Response [200]>
```

Don't override `Authorization` header when contents are bearer token (or any other token) #3929

Open **tomvlk** opened this issue on Mar 19, 2017 · 17 comments

danrue commented on Nov 1, 2018

Coming here after spending several hours debugging an issue which ended up being the presence of a ~/.netrc file. This behavior violates POLA and should be explicitly enabled rather than enabled by default.

8

bors-ltd commented on Apr 23, 2019

I lost half a day because I could not log to production any more, and I couldn't find the issue in our infrastructure. Found out it was because I stored my password in `~/.netrc` and requests read it and added an `Authorization` header when I was using a `Bearer` instead, and got rejected from the server.

It should only happen with an explicit `BasicAuth()`.

5