How to write a Python script authenticating with Jira via OAuth

If you are writing a script that interacts with Jira through a REST API, you should authenticate using an OAuth token, rather than an embedded username /password. Here we describe one way to do the 'oauth dance' to generate a trusted token using Python 3 - specifically the `jirashell` utility from the `jira` Python package. `jirashell` then forms a useful basis for a Python script. Our example script uses OAuth to call an undocumented REST API for querying license data.

These instructions no longer work for me on tested combinations {Ubuntu 18.04 + Python 3.6.9}, { Ubuntu 20.04 + Python 3.8.5}. The `jirashell` command should print a URL, but instead returns:

```
'oauth_token'
```


Establishing OAuth trust

Install Python 3

Running `python3` or `python --version` should show Python 3.x.

Create a venv

```
mkdir jira-oauth
cd jira-oauth
python3 -m venv venv
. venv/bin/activate
```

Install Python libraries

```
pip3 install -U pip   # upgrade pip to avoid "No module named 'setuptools_rust'" error
pip3 install jira ipython
```

If you get an error:

```
Collecting cryptography>=2.0 (from SecretStorage>=3.2; sys_platform == "linux"->keyring-
>jira)
  Downloading https://files.pythonhosted.org/packages/9b/77
/461087a514d2e8ece1c975d8216bc03f7048e6090c5166bc34115afdaa53/cryptography-3.4.7.tar.gz (546kB)
    100% || 552kB 2.8MB/s
    Complete output from command python setup.py egg_info:

            ============================DEBUG ASSISTANCE==========================
            If you are seeing an error here please try the following to
            successfully install cryptography:

            Upgrade to the latest pip and try again. This will fix errors for most
            users. See: https://pip.pypa.io/en/stable/installing/#upgrading-pip
            ============================DEBUG ASSISTANCE==========================

    Traceback (most recent call last):
      File "<string>", line 1, in <module>
      File "/tmp/pip-build-xs9c9nwd/cryptography/setup.py", line 14, in <module>
        from setuptools_rust import RustExtension
    ModuleNotFoundError: No module named 'setuptools_rust'

    ----------------------------------------
Command "python setup.py egg_info" failed with error code 1 in /tmp/pip-build-xs9c9nwd/cryptography/
```

then **pip3 install -U pip** should fix it.

Generate an RSA public key

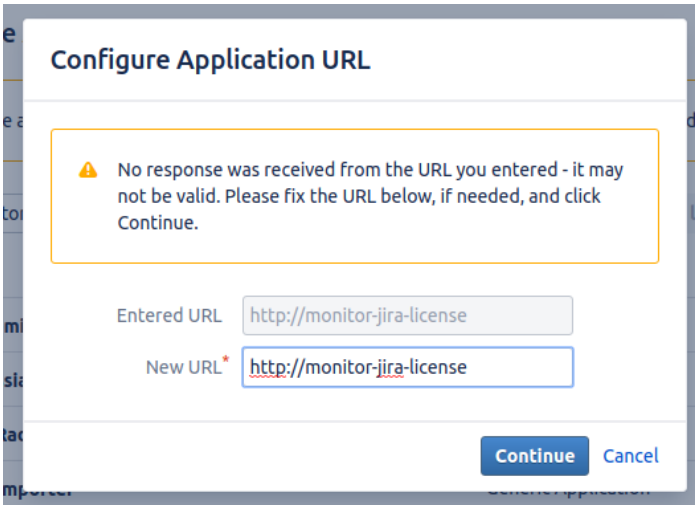
```
openssl genrsa -out rsa.pem 2048
openssl rsa -in rsa.pem -pubout -out rsa.pub
```

Create an application link

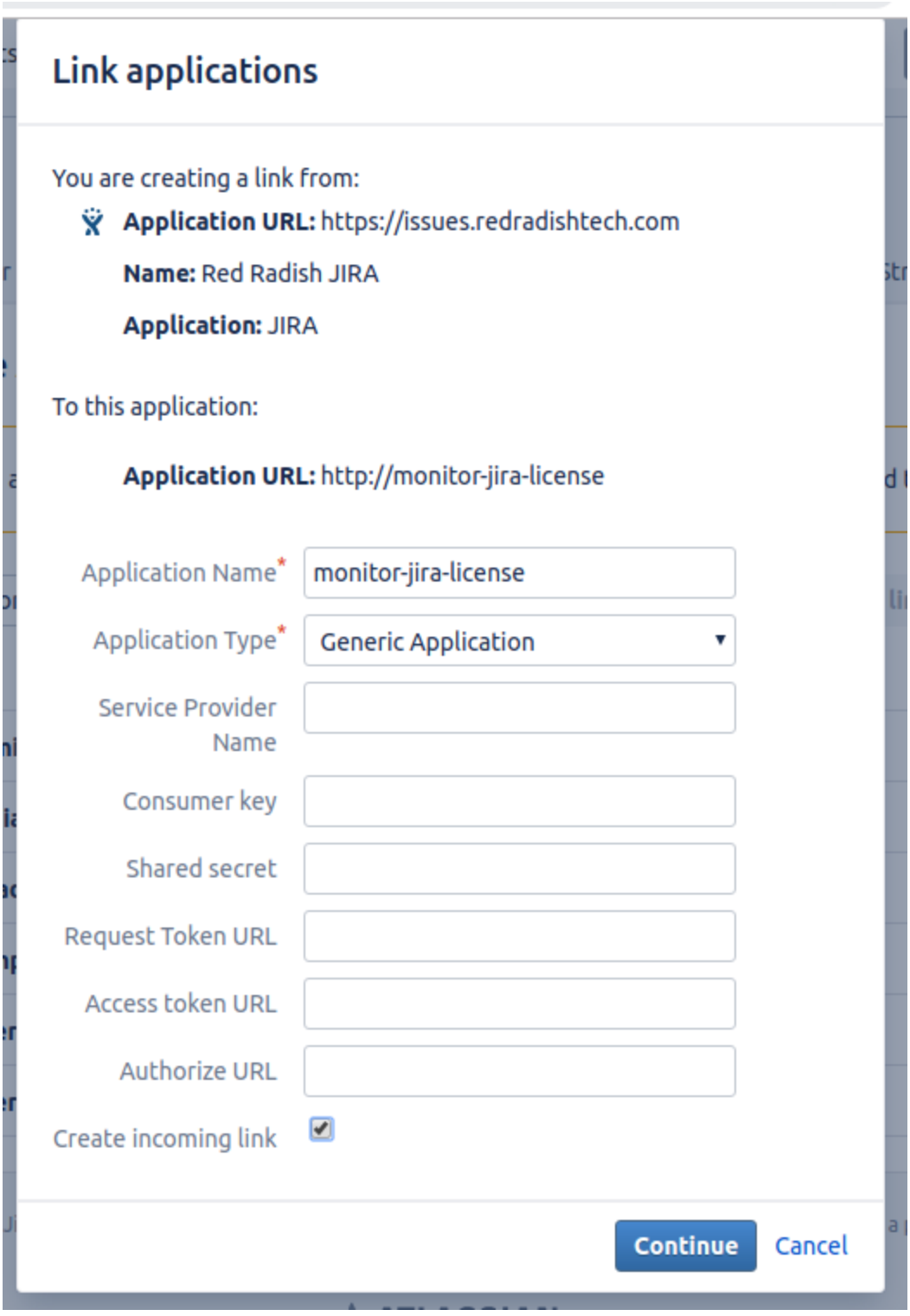
In Jira, create an applink. Applinks normally connect to other HTTP apps, but in this case our OAuth client doesn't have a URL, so use a fake one.

I originally created these instructions when creating an OAuth token for a Nagios Jira license monitor, hence the token I use is **monitor-jira-license** , and the fake URL is http://monitor-jira-license:

Jira will complain, but just click Continue:



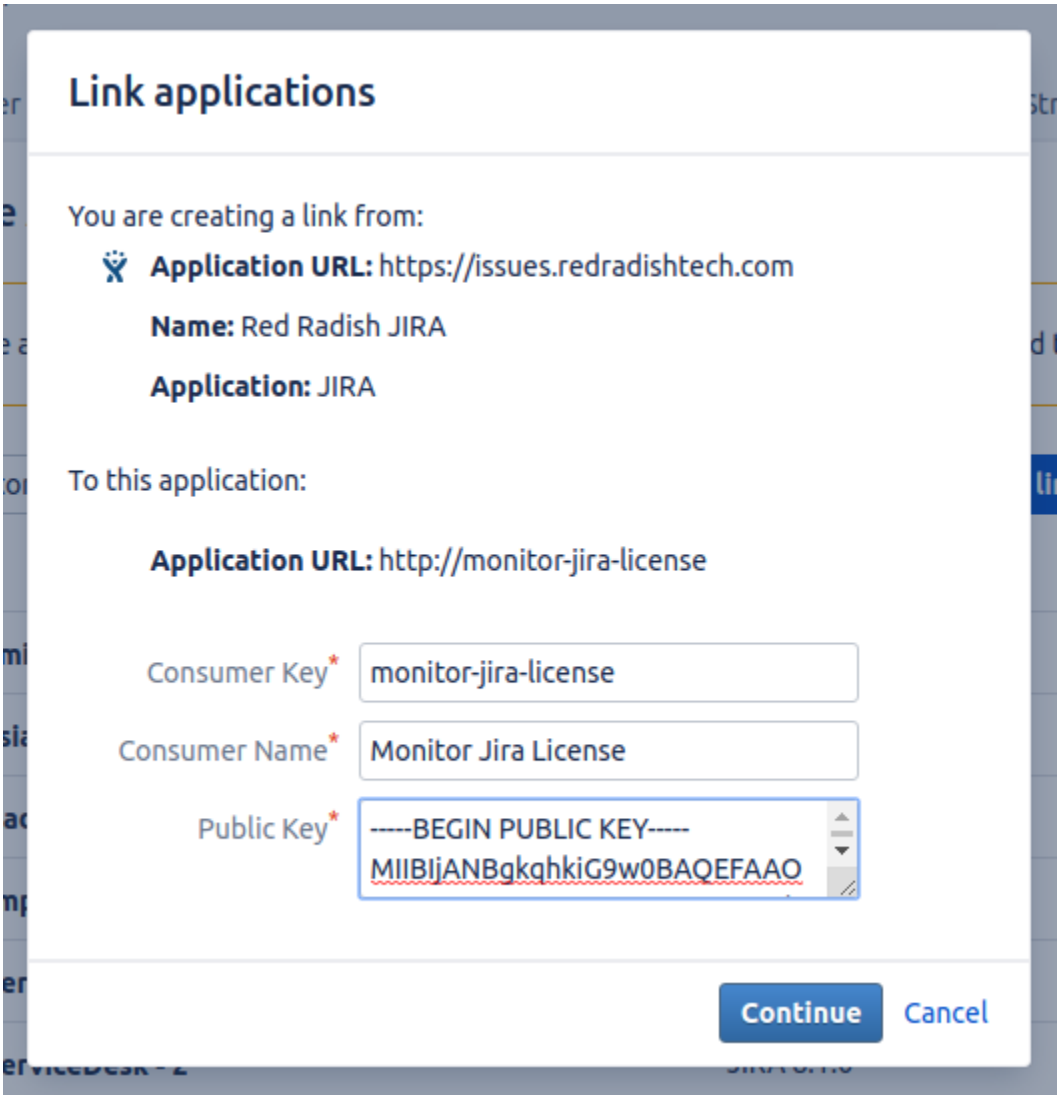
On the next page, enter 'monitor-jira-license' as the Application Name. Leave other fields blank. Check the 'Create incoming link' checkbox:



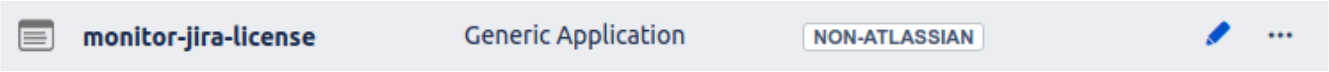
On the next page, fill in:

Field	Value	Notes
Consumer Key	monitor-jira-license	this key will be used in the script
Consumer Name	Monitor Jira License	any descriptive text

Public key	*contents of rsa.pub*	



Click 'Continue', and your application link will be created.



OAuth dance

Now from your terminal, do the OAuth dance with your Jira installation:

```
BROWSER='echo %s' jirashell --server https://issues.redradishtech.com --consumer-key monitor-jira-license --key-cert rsa.pem --oauth-dance
```

If, instead of printing a URL, the jirashell command just prints:

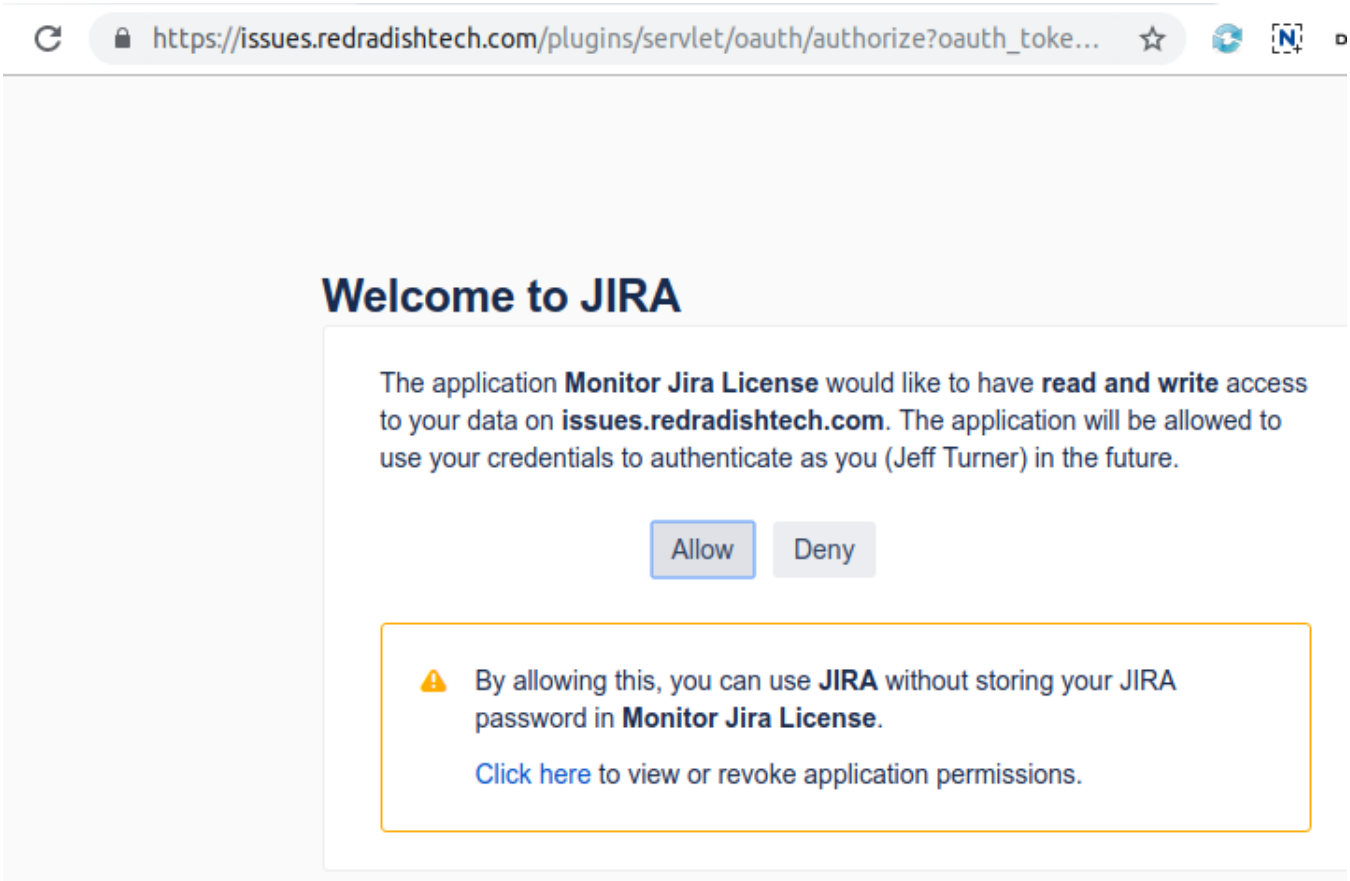
```
'oauth_token'
```

That means your `consumer-key` does not exist in Jira (perhaps you copy & pasted the example without substituting yours?). This is an error reporting bug in jirashell.

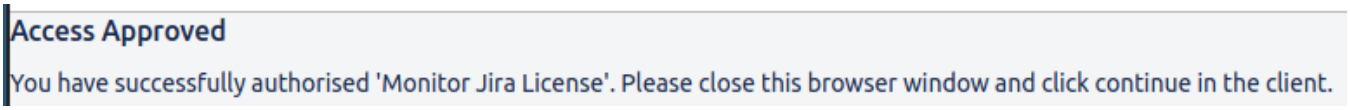
Jirashell would normally try to launch your preferred web browser, using the webbrowser library. By setting the BROWSER env variable, we tell Python not to bother, and just print the URL for us to manually cut & paste. This is required for server environments, where `lynx` isn't able to deal with Jira's Javascript.

This should print a URL.

At this point you need to decide which JIRA user you want to grant OAuth access as. For most scripts you should create a dedicated JIRA role account with reduced privileges. **Log out and back in to JIRA as that user**, (or use switchuser.jsp) then open the link:



Click 'Allow' in the Browser window:



If you have

After the URL, your terminal also should have displayed:

```
Your browser is opening the OAuth authorization for this client session.
Have you authorized this program to connect on your behalf to https://issues.redradishtech.com? (y/n)
```

Press 'y'.

The jirashell command now proceeds to launch an IPython session:

```
<JIRA Shell 2.0.0 (https://issues.redradishtech.com)>

*** JIRA shell active; client is in 'jira'. Press Ctrl-D to exit.

In [1]:
```

Type `oauth` to print the OAuth object:

Now press ctrl-d to exit.

Test your OAuth token

Now embed the 'consumer_key', 'access_token' and 'access_token_secret' values you saw above into a new jirashell command:

```
jirashell --server https://issues.redradishtech.com --consumer-key monitor-jira-license --access-token
A56FItjuH3jfcCs4aYS57gzXnAPXk2Zt --access-token-secret t8JaIJGSsxqZLRQoDQbQYm9f761zgvPs --key-cert rsa.pem  <<<
'jira.server_info()'
```

If successful, the `jira.server_info()` command piped to stdin should succeed:

```
<JIRA Shell 2.0.0 (https://issues.redradishtech.com)>

*** JIRA shell active; client is in 'jira'. Press Ctrl-D to exit.

In [1]: Out[1]:
{'baseUrl': 'https://issues.redradishtech.com',
 'version': '7.13.0',
 'versionNumbers': [7, 13, 0],
 'deploymentType': 'Server',
 'buildNumber': 713000,
 'buildDate': '2018-11-28T00:00:00.000+1100',
 'scmInfo': 'fbf406879436de2f3fb1cfa09c7fa556fb79615a',
 'serverTitle': 'Red Radish JIRA'}

In [2]: Do you really want to exit ([y]/n)?
```

You now have the three things you need for your script: the token, the token secret, and `rsa.pub` private key.

Using `jirashell` as a base for your script

If your script is Python, you can use `jirashell` as a library to handle all the ugly command-line parsing. In my case:

```
$ cp venv/bin/jirashell check-jira-license
$ vim check-jira-license   # Make changes
$ cat check-jira-license
#!/home/jturner/src/redradish/nagios-jira-license/venv/bin/python3

# -*- coding: utf-8 -*-
import re
import sys

from jira.jirashell import get_config, JIRA

def main():
    options, basic_auth, oauth = get_config()

    jira = JIRA(options=options, oauth=oauth)

    print(jira.server_info())

if __name__ == '__main__':
    sys.exit(main())
```

This command can then be invoked using the same command-line flags as `jirashell` :

```
./check-jira-license --server https://issues.redradishtech.com --consumer-key monitor-jira-license --access-
token kLYKeT0g9EiJDDmqlxQTH9VjRs2fpFS6 --access-token-secret snhWUlGQmzLu6I9ju1aQGNjulQQPT1lz --key-cert rsa.pem
```

Some JIRA REST calls are not wrapped in the Python JIRA library. For those, you can use the OAuth credentials with the `requests` library directly, as follows:

```
#!/home/jturner/src/redradish/nagios-jira-license/venv/bin/python3

# -*- coding: utf-8 -*-
import re
import sys

from jira.jirashell import get_config, JIRA
import requests

def getlicensecounts(options, jira):
    url=options['server'] + '/rest/plugins/applications/1.0/installed/jira-software'
    response = requests.get(url,  auth=jira._session.auth)
    responsejson = response.json()
    return (responsejson['accessDetails']['activeUserCount'], responsejson['accessDetails']
['licensedUserCount'])

def main():
    options, basic_auth, oauth = get_config()

    jira = JIRA(options=options, oauth=oauth)
    activecount, totalcount = getlicensecounts(options, jira)
    print(f"Using {activecount} of {totalcount} license slots")

if __name__ == '__main__':
    sys.exit(main())
```

You can invoke non-REST (`/secure/admin/*`) URLs with OAuth credentials too, but Jira's "websudo" authentication will demand a password, rendering OAuth useless.