# Migrating JIRA users to LDAP, preserving passwords

JIRA gives you the choice of storing user records internally, or delegating to an external 'User Directory' like Active Directory, LDAP or Atlassian Crowd.

Many smaller orgs start off with internal user records, but later want to migrate users to LDAP for ease of management, or to allow authentication with non-Atlassian LDAP-aware systems.

Generating LDAP (LDIF) records from JIRA's cwd_* tables is not hard, but how about password hashes?

On this page we'll describe how to convert JIRA credential hashes:

```
{PKCS5S2}U48fu6LonjKCk0VmHPsgLrKf1/i1o/wxLXblOTa6P8eXvvJTU4iRb0fpRlO3xA0J
```

into a format understandable by OpenLDAP (with the `pw-pbkdf2` module loaded):

```
{PBKDF2}10000$cjsPF6FcSW9CDwmpREtZog$qWi06T.6SSapuTtDsFn/2DPacsc
```

This will let you migrate user records from Jira into LDAP without forcing everyone to reset their password.

> Argh! Something is buggy in hash conversion process. It works for some passwords but not for others ('hunter2' in particular).
>
> I never ended up using this beyond testing, so don't have inclination to debug. I've left it online for all the incidental information provided.

Those familiar with JIRA's database will know about the `cwd_user` table, where JIRA stores user data:

```
redradish_jira=> select * from cwd_user where user_name='jturner';
[ RECORD 1 ]
 id                  10000
 directory_id        1
 user_name           jturner
 lower_user_name     jturner
 active              1
 created_date        2013-09-02 18:14:34.078712+10
 updated_date        2018-02-23 10:33:48.481+11
 first_name          Jeff
 lower_first_name    jeff
 last_name           Turner
 lower_last_name     turner
 display_name        Jeff Turner
 lower_display_name  jeff turner
 email_address       jeff@redradishtech.com
 lower_email_address jeff@redradishtech.com
 credential          {PKCS5S2}U48fu6LonjKCk0VmHPsgLrKf1/i1o/wxLXblOTa6P8eXvvJTU4iRb0fpRlO3xA0J
 deleted_externally
 external_id         a330dede-18f8-4745-ac8d-d2ec2bcabedc
```

## Atlassian's PKCS5S2 format

What exactly is that PKCS5S2 format JIRA uses for password hashes?

'**PKCS5S2**' refers to "PKCS #5: Password-Based Cryptography Specification Version 2.0", a document available in RFC form which provides "recommendations for the implementation of password-based cryptography" . The recommendations include the use of the **PBKDF2** 'key derivation function', of which HMAC-SHA-1 is an example.

The format is succinctly explained in the `passlib.hash.atlassian_pbkdf2_sha1` Python library's docs:

- generate a random 16-byte salt
- feeds the salt plus password into our PBKDF2 function, which applies a hash (HMAC-SHA1) 10,000 times, yielding a a 32-byte hash
- concatenates salt and hash, and base64-encodes them

Incidentally you can generate such a hash using Python:

```
$ sudo pip3 install passlib
$ python3 -c 'from passlib.hash import atlassian_pbkdf2_sha1; print(atlassian_pbkdf2_sha1.hash("hunter2"));'
{PKCS5S2}sFaqFaJUijGG0FqLUQrhPOEXrxB7jrXI7lzkPstbM3bhPq7x8rSS+Q3NtSduIgwt
```

If you have a commercial Jira license, you can also download the source at https://my.atlassian.com and take a look (unpack `dependencySources/atlassian-password-encoder-*-sources.jar` and look at DefaultPasswordEncoder and PKCS5S2PasswordHashGenerator).

So, easy enough. Let's unpack our sample password:

```
$ credential='{PKCS5S2}U48fu6LonjKCk0VmHPsgLrKf1/i1o/wxLXblOTa6P8eXvvJTU4iRb0fpRlO3xA0J'
$ credential="${credential#'{PKCS5S2}'}"                                                    # Chop off the
identifier
$ echo $credential
U48fu6LonjKCk0VmHPsgLrKf1/i1o/wxLXblOTa6P8eXvvJTU4iRb0fpRlO3xA0J
$ echo -n "$credential" | base64 -d | xxd
00000000: 538f 1fbb a2e8 9e32 8293 4566 1cfb 202e  S......2..Ef.. .
00000010: b29f d7f8 b5a3 fc31 2d76 e539 36ba 3fc7  .......1-v.96.?.
00000020: 97be f253 5388 916f 47e9 4653 b7c4 0d09  ...SS..oG.FS....
```

The first 16 bytes is our salt, and the remainder is our hash:

```
$ salt="$(echo -n "$credential" | base64 -d | head -c16)"
$ hash="$(echo -n "$credential" | base64 -d | tail -c32)"
```

## OpenLDAP's PBKDF2 Support

OpenLDAP supports PBKDF2 with the help of a module. Here is how to generate a hash from the command-line:

```
slappasswd -o module-load=pw-pbkdf2.la -h {PBKDF2} -s hunter2
{PBKDF2}10000$wf6MXP0w8pxfQXKqDWCK1g$O3Vb3KDkFcmTqBCZU0w97XlELFc
```

The format is:

```
{PBKDF2}<Iteration>$<Adapted Base64 Salt>$<Adapted Base64 DK>
```

Although Atlassian's {PKCS5S2} and OpenLDAP's {BPKDF2} are really the same thing, the format is a bit different. Our job is to convert from Atlassian's to OpenLDAP's.

This is not hard. Look at OpenLDAP's format again:

```
{PBKDF2}<Iteration>$<Adapted Base64 Salt>$<Adapted Base64 DK>
```

We know the iteration count (10000). We know the salt. We know the hash (derived key). We just need to reorder the elements.

Also, what is "adapted Base64"? Per the passlib docs it is just a shortened base64 format which trims the padding (appearing as '=' at the end of base64-encoded strings), and uses '.' characters instead of '+'.We can define this as a bash function:

```
$ ab64encode() { python3 -c 'import sys; from passlib.utils.binary import *; print(ab64_encode(sys.stdin.buffer.
read()).decode("utf-8"))'; }
$ echo foo | base64        # regular base64
Zm9vCg==
$ echo foo | ab64encode    # adapted base64
Zm9vCg
```

Now we have everything we need to write a conversion function:

```
function atlassian_to_pbkdf2()
{
  ab64encode() { python3 -c 'import sys; from passlib.utils.binary import *; print(ab64_encode(sys.stdin.buffer.
read()).decode("utf-8"))'; }
  local credential="$1"
  credential="${credential#'{PKCS5S2}'}"
  salt="$(echo -n "$credential" | base64 -d | head -c16 | ab64encode)"
  hash="$(echo -n "$credential" | base64 -d | tail -c32 | ab64encode)"
  printf "Salt: %s\n" "$salt"
  printf "Hash: %s\n" "$hash"
  printf "{PBKDF2}%d$%s$%s" 10000 "$salt" "$hash" | head -c64
  echo
}
```

or in Python if you prefer:

**atlassian_to_pbkdf2.py**

```
#!/usr/bin/env python3
# Converts Atlassian's password format:
#
# to OpenLDAP's format:
# {PBKDF2}<Iteration>$<Adapted Base64 Salt>$<Adapted Base64 DK>

import sys
from passlib.utils.binary import b64decode
from passlib.utils.binary import ab64_encode

credential = sys.argv[1]   # {PKCS5S2}U48fu6LonjKCk0VmHPsgLrKf1/i1o/wxLXblOTa6P8eXvvJTU4iRb0fpRlO3xA0J
#credential="{PKCS5S2}U48fu6LonjKCk0VmHPsgLrKf1/i1o/wxLXblOTa6P8eXvvJTU4iRb0fpRlO3xA0J"
credential = credential[9:]                             # U48fu6LonjKCk0VmHPsgLrKf1/i1o
/wxLXblOTa6P8eXvvJTU4iRb0fpRlO3xA0J
b64decode(credential)
salt = ab64_encode( b64decode(credential)[0:16] ).decode('ascii')
hash = ab64_encode( b64decode(credential)[16:48] ).decode('ascii')
final=f"{{PBKDF2}}10000${salt}${hash}"
print(final[:64])
```

A sample run:

```
$ atlassian_to_pbkdf2 {PKCS5S2}U48fu6LonjKCk0VmHPsgLrKf1/i1o/wxLXblOTa6P8eXvvJTU4iRb0fpRlO3xA0J
{PBKDF2}10000$U48fu6LonjKCk0VmHPsgLg$sp/X.LWj/DEtduU5Nro/x5e.8lN
```

Testing that this is correct is a bit tricky. Per the advice for OpenLDAP PBKDF2 page, the best way is probably to set this password in your `/etc/ldap/slapd.conf`:

```
moduleload      pw-pbkdf2.so
...
rootdn          "cn=admin,dc=redradishtech,dc=com"
rootpw          {PBKDF2}10000$U48fu6LonjKCk0VmHPsgLg$sp/X.LWj/DEtduU5Nro/x5e.8lN
```

## What about {SHA} password hashes?

Up till 2013 JIRA (and Crowd) used the 'atlassian-sha1' scheme, which was actually unsalted sha512 (see ~~CWD-1137~~ - Default to sha1 hashes rather than the infrequently implemented atlassian-sha1 CLOSED ). I implemented sha512 support for OpenLDAP to support this (see https://git.openldap.org/openldap/openldap/-/tree/master/contrib/slapd-modules/passwd/sha2).