Two-Factor Authentication

Passwords, everyone loves to hate them and still, in the era of digital certificates, fingerprints, and voice recognition, we use them on a daily basis and want users to memorize tens of different complex passwords. So they cheat and passwords get reused, written down on a piece of paper, you name it. Not that service providers always get this right either.

And any time we use a password, we run the risk of it being intercepted, and at that point it's no longer secret and needs to be changed. Still, passwords have one important advantage over most other forms of authentication: as a simple string they can be used anywhere. Everyone understands a login prompt and it tends to be rather hard to make anything else work for everyone.

One-time passwords represent a modest improvement being relatively unobtrusive for the user while anyone who sees the password should be unable to do anything with it. The user has little extra work to do, and chances are they already carry a smartphone with them. But how do we get existing applications to support them without extensive modification?

For applications that use OpenLDAP for authentication, the answer is easy: enable the `otp_2fa` overlay and share another secret with the user. It can be as simple as letting them scan a QR code.

Getting Started

Symas OpenLDAP Gold now includes a module that lets any application that authenticates through LDAP to work with time-based and counter-based one-time passwords.

Module

The Two-factor Authentication module is standard in versions of Symas OpenLDAP Gold and Silver starting with release 2.4.46.1.

Configuration

Load the `otp_2fa` module:

```
ldapmodify -x –H ldap:/// -D cn=config –W
dn: cn=module{0},cn=config
changetype: modify
add: olcModuleLoad
olcModuleLoad: otp_2fa.la
```

Add the `otp_2fa` overlay to the definition of each database that will allow OTP authentication

```
ldapadd -x –H ldap:/// -D cn=config –W
dn: olcOverlay=otp_2fa,olcDatabase={X}YYY,cn=config
objectClass: olcOverlayConfig
```

Provisioning the Common OTP Parameters

To let the overlay know how to generate the one-time passwords, we store the common HOTP and/or TOTP parameters. Usually, it is the organization or group entry.

For TOTP we care about the password length, hash algorithm, time step size and a grace time window. Popular sites usually choose 6-character passwords with SHA1 hash updated every 30 seconds with a window of over a minute (three time steps).

```
ldapmodify -x –H ldap:/// -D o=example -W
dn: ou=people,dc=example,dc=com
changetype: modify
add: objectClass
objectClass: oathTOTPParams
-
add: oathOTPLength
oathOTPLength: 6
-
add: oathHMACAlgorithm
# choose SHA1, algorithm OIDs are specified in RFC 8018
oathHMACAlgorithm: 1.2.840.113549.2.7
-
add: oathTOTPTimeStepPeriod
oathTOTPTimeStepPeriod: 30
-
add: oathTOTPTimeStepWindow
oathTOTPTimeStepWindow: 3
```

Alternatively, for HOTP we care about the password length, hash algorithm and look-ahead. Popular sites usually choose 6-character passwords with SHA1 hash and a look-ahead of 3 passwords.

```
ldapmodify -x –H ldap:/// -D cn=manager,cn=people,o=example -W
dn: ou=people,dc=example,dc=com
changetype: modify
add: objectClass
objectClass: oathHOTPParams
-
add: oathOTPLength
oathOTPLength: 6
-
add: oathHMACAlgorithm
# choose SHA1, algorithm OIDs are specified in RFC 8018
oathHMACAlgorithm: 1.2.840.113549.2.7
-
add: oathHOTPLookAhead
oathHOTPLookAhead: 3
```

Now that the server knows how to use TOTP, we can let our user set things up. They start with generating a random shared secret that's at least 20 bytes long:

```
touch sharedkey

chmod 600 sharedkey

openssl rand 20 > sharedkey

base32 sharedkey
```

Example output:

N6CIHPYDGHI6QC4ZV7Q3S3FXA22BZOCY

They then add the secret as a new token. A token can also be shared between accounts.

Example for TOTP:

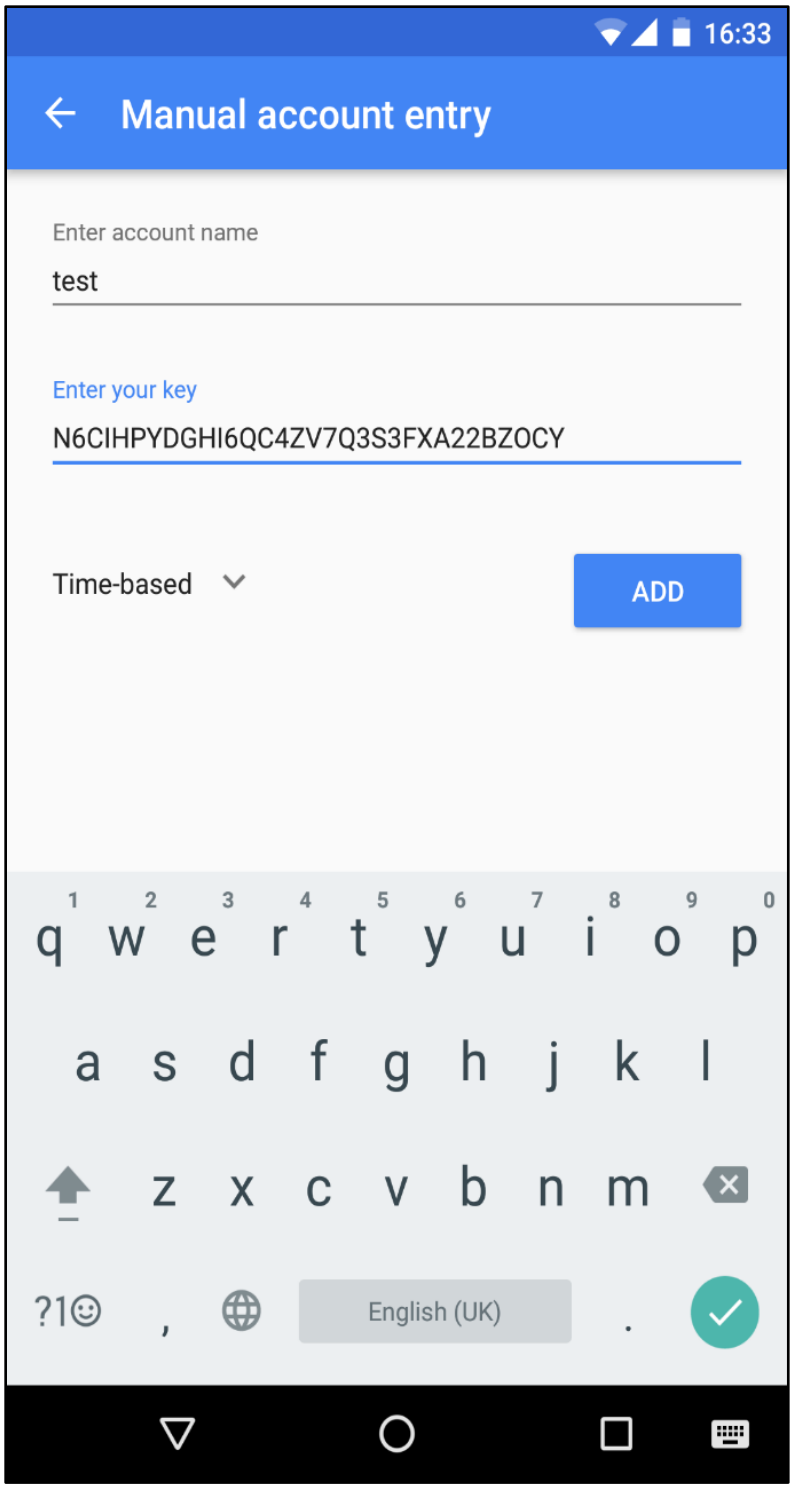
```
ldapmodify -x –H ldap:/// -D cn=user,ou=people,dc=example,dc=com –W
dn: cn=user,ou=people,dc=example,dc=com
changetype: modify
add: objectClass
objectClass: oathTOTPToken
-
add: oathTOTPParams
oathTOTPParams: ou=people,dc=example,dc=com
-
add: oathSecret
oathSecret:< file://sharedkey
-
add: objectClass
objectClass: oathTOTPUser
-
add: oathTOTPToken
oathTOTPToken: cn=user,ou=people,dc=example,dc=com
```

Example for HOTP:

```
ldapmodify -x –H ldap:/// -D cn=user,ou=people,dc=example,dc=com –W
dn: cn=user,ou=people,dc=example,dc=com
changetype: modify
add: objectClass
objectClass: oathHOTPToken
-
add: oathHOTPParams
oathHOTPParams: ou=people,dc=example,dc=com
-
add: oathSecret
oathSecret:< file://sharedkey
-
add: objectClass
objectClass: oathHOTPUser
-
add: oathHOTPToken
oathHOTPToken: cn=user,ou=people,dc=example,dc=com
```

The user now adds this shared key to their authenticator.

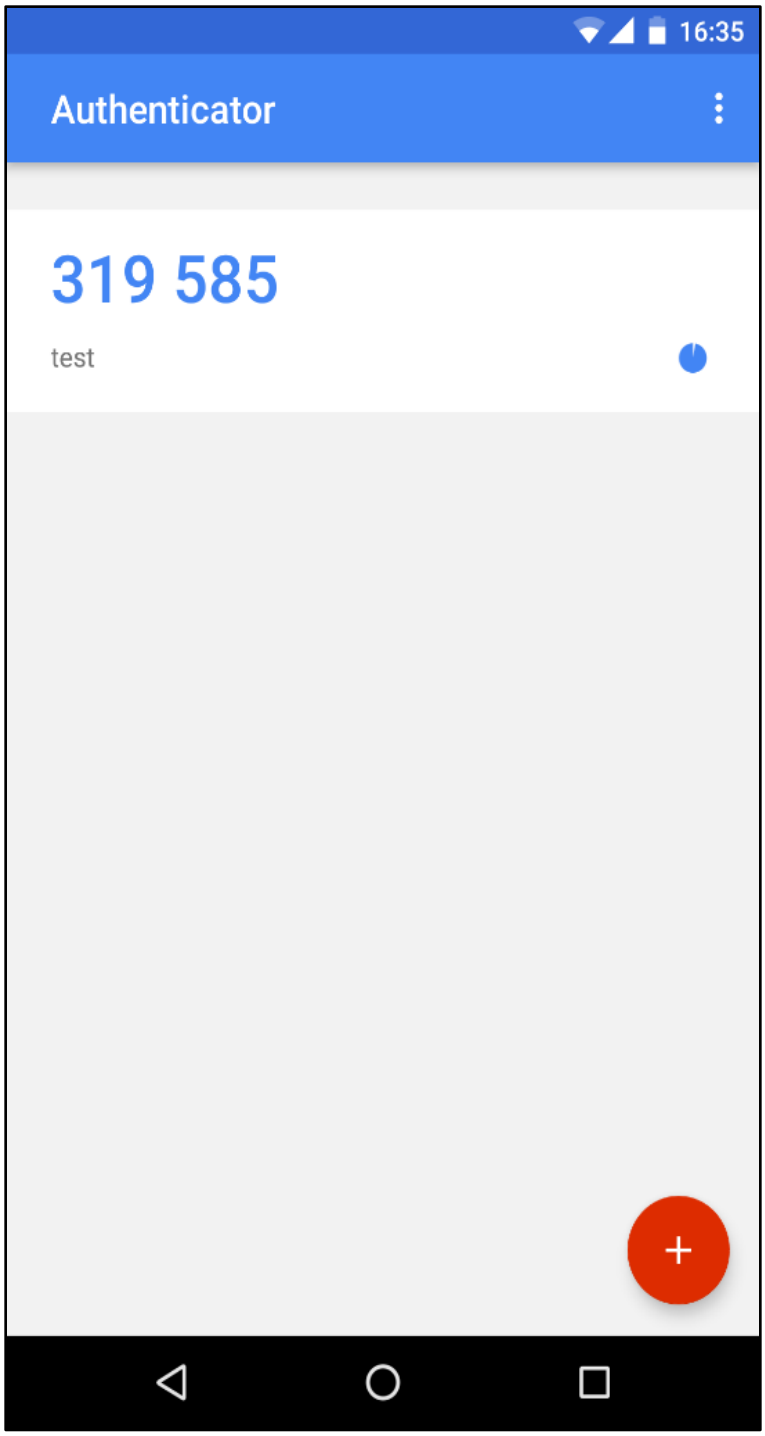
In the Google Authenticator app, they would select 'Enter a provided key' and then type the generated string.



Showtime

Now it's time to test.

That's it! It's now trivial to log into the system: whenever you get asked to enter your password, check with your device and enter your password, immediately followed by the six-digit code that appears on the screen:



```
ldapwhoami -x -H ldap://localhost -D 'cn=user,dc=example,dc=com' -W
Enter LDAP Password:
dn:cn=user,ou=people,dc=example,dc=com
```

If your systems rely on your server for user authentication and use LDAP Binds, you are set. Once a user has a shared key set in the database, they can use the token to log in alongside their usual password. And just like they remember their password, they make sure they retain control over their device.

Final Notes

If you decide to use this in production, you might want to make it easier for users to manage their keys. Many authenticator apps support reading the key from a QR code, so rendering the key in a QR code on a web page becomes a handy way to deliver it. In modern web systems rendering QR codes is easy with generators written in node.js, such as [qr-image (https://github.com/alexeyten/qr-image).

This is a based one-time password scheme, and as such you will have some trouble using it as the only method of authentication when the credentials have to be reused, or if there is a delay between when the user enters the password and when authentication happens. In these circumstances, switching to proxy authorization once you have authenticated your user might be an option worth investigating.