Configure Time-Based One-Time Passwords ((T)OTP)

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 producers 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: turn on time-based one-time password authentication.

Getting started

Symas OpenLDAP now includes a password module that lets any application that authenticates through LDAP to work with time-based one-time passwords (https://tools.ietf.org/html/rfc6238).

Module

The TOTP module is standard in versions of Symas OpenLDAP Gold and Silver starting with release 2.4.44.1.

Also, starting with release 2.4.xx.xx, the packages also come with a script that sets up an example database running all the below steps for you. You can find that script in `/opt/symas/etc/openldap/example-totp.sh`.

Configuration

1. Load the TOTP module:

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

2. Set the default password hash to TOTP1:

```
ldapmodify -x -D cn=config -W -H ldap://localhost
dn: cn=config
changetype: modify
replace: olcPasswordHash
```

```
olcPasswordHash: {TOTP1}
```

3. Add the TOTP overlay to the definition of each database that will allow TOTP authentication To enforce the passwords are valid only once, the module exposes an overlay that enables TOTP functionality on the database. For each database that you will be storing users with TOTP set up, enable the overlay:

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

Setting the TOTP Secret

Now that the server knows how to use TOTP, we can let our user set things up.

1. 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 N6CIHPYDGHI6QC4ZV7Q3S3FXA22BZOCY
```

2. Set the sharedkey as their password. Having already set the default scheme to 'TOTP1' above, we made sure that server will treat it as a TOTP secret.

```
ldappasswd -T sharedkey -x -H ldap://localhost –D
cn=user,dc=example,dc=com -W
```

Or, if we are acting on behalf of the user (they might not have a password yet), we can tell ldappasswd whose password it is that we are changing:

```
ldappasswd -T sharedkey -x -H ldap://localhost -D dc=example,dc=com -W
\ cn=user,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. To log into the system: whenever you get asked to enter your password, check with your device and enter 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,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 just like they would their 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 (https://github.com/google/google-authenticator/wiki/Key-Uri-Format), 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 time-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 disconnect between the time the user enters the password and when authentication happens. In these circumstances, switching to SASL proxied authentication once you have authenticated your user might be an option worth investigating.

The TOTP overlay depends on the authTimestamp attribute to make sure the password cannot be reused, this makes it impossible to use the lastbind overlay on the same database.

There is a limitation that makes this TOTP implementation behave slightly different from others. The authenticator's clock might drift from the server's own and it might take some time for the user to enter the one-time password. For these reasons, servers can usually correct a short time difference and record the perceived drift. These resynchronization capabilities are not available in this version, and while a 30 second window is usually plenty of time, any time drift between the device and server might make it more difficult for the user to log in. Lifting this limitation is on our roadmap, so stay tuned!

Complete cn=config Example

```
dn: cn=config
objectClass: olcGlobal

dn: cn=module{0},cn=config
objectClass: olcModuleList
olcModulePath: /opt/symas/lib64/openldap
olcModuleLoad: back_mdb.la
olcModuleLoad: pw-totp.la

dn: cn=schema,cn=config
objectClass: olcSchemaConfig

dn: cn={0}core,cn=schema,cn=config
objectClass: olcSchemaConfig
olcAttributeTypes: ( 2.5.4.4 NAME ( 'sn' 'surname' )
  DESC 'RFC2256: last (family) name(s) for which the entity is known by'
  SUP name )
olcAttributeTypes: ( 2.5.4.20 NAME 'telephoneNumber'
  DESC 'RFC2256: Telephone Number'
  EQUALITY telephoneNumberMatch
  SUBSTR telephoneNumberSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.50{32} )
olcObjectClasses: ( 2.5.6.6 NAME 'person'
  DESC 'RFC2256: a person'
  SUP top STRUCTURAL
  MUST ( sn $ cn )
  MAY ( userPassword $ telephoneNumber $ seeAlso $ description ) )
olcAttributeTypes: ( 0.9.2342.19200300.100.1.25
  NAME ( 'dc' 'domainComponent' )
  DESC 'RFC1274/2247: domain component'
  EQUALITY caseIgnoreIA5Match
  SUBSTR caseIgnoreIA5SubstringsMatch
```

```
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 SINGLE-VALUE )
olcObjectClasses: ( 1.3.6.1.4.1.1466.344 NAME 'dcObject'
  DESC 'RFC2247: domain component object'
  SUP top AUXILIARY MUST dc )

dn: olcDatabase={0}config,cn=config
objectClass: olcDatabaseConfig
olcRootDN: cn=config
olcRootPW: secret

dn: olcDatabase={1}mdb,cn=config
objectClass: olcMdbConfig
olcDbDirectory: /var/symas/openldap-data/example
olcSuffix: dc=example,dc=com
olcRootDN: dc=example,dc=com
olcRootPW: secret

dn: olcOverlay=totp,olcDatabase={1}mdb,cn=config
objectClass: olcOverlayConfig
```

Complete slapd.conf Example

```
include /opt/symas/etc/openldap/schema/core.schema

modulepath /opt/symas/lib64/openldap
moduleload back_mdb.la
moduleload pw-totp.la

password-hash {TOTP1}

database mdb
directory /var/symas/openldap-data/example
suffix "dc=example,dc=com"
rootdn "dc=example,dc=com"
rootpw secret

overlay totp
```