Safer printing with OL Connect Send

A peek at the new OL Connect Send feature that will allow servers to check if the client is authorized to submit a print job. With this new OLCS client, servers can now decide which print jobs are allowed.

Authorization or Authentication

API keys provide a way for authorization, in other words, a way for the server to decide if the client is *allowed* to transfer a job. This is different from authentication to establish identity, to determine *who* is submitting a job. The difference between these two becomes very clear when multiple people use the same API key to print. When identity is required, it is best to have the user authenticate themselves, instead of relying on a piece of software to do so on their behalf. This is why we have focused on authorization for the OLCS client. Authentication can be handled through the browser with the interactive url.

Authorization with API keys

To prevent unauthorized OLCS clients from submitting jobs, we have adopted a simple and well-known approach: API keys. You can see these used by email service providers such as SendGrid, and Mailjet, and it's also what TIE Kinetix use to receive data.

Basically, an API key is just a string that is not easy to guess that the client will send along when submitting a job. It's a lot like a password, but, since it doesn't have to be remembered or typed by a person, it can be a *lot* longer than a password would typically be.

Sample API key

```
OL.bd34450fa927201fb4c506beaf5d6fbb.77a051605ed181bab91fd725d40fd8ad4097538f65c5d29dbda2f5224cdfd54f
```

The user who prints jobs with OL Connect Send, doesn't have to do anything special to make the API work; it is configured when the OLCS client is installed. When installing, it can be put in the ini file of the installer or entered in the installation wizard. The client will then use it in the background to talk to the server.

The API key goes in a standard http header

```
Authorization: Bearer OL.bd34450fa927201fb4c506beaf5d6fbb.77a051605ed181bab91fd725d40fd8ad409753...
```

On the server side, the process receiving OLCS print jobs must check if an API key was supplied. We have a sample Workflow process that shows how this can be done. By performing this check before the job is handed to the Job Processor task of OLCS, we can prevent deducting credits for unauthorized job transfers. The job data will not be transferred to the server, because an unauthorized client will break off the transfer even before it gets to sending the job data.

If an API key is suspected to be misused or leaked, it must be revoked to block any (further misuse. Revoking can be as simple as deleting the suspect key from where it is stored. The legitimate users then of course have to receive a new API key. If there is no urgent reason for immediate action, it's also possible to issue a new key before revoking the old one.

How API keys are created, revoked, and where they are stored, is left open. Service providers that want to integrate a Connect Send based solution into an existing portal and platform, can create their own API key management on that platform. We have however, created a sample API key management in Connect Workflow, together with a WordPress plugin that lets users manage API keys in the WordPress admin interface.

The API key is only used by the OLCS client to transfer print jobs to the server, and not for interaction. The API key is not necessary for interaction, because here the user is involved, and can do a normal login. In addition, the interactive url probably doesn't even point to Connect Workflow, but to some portal. Lastly, it's technically not feasible to use an API key for interaction, because this takes place from the browser instead of the OLCS client.

API key strategy

The basic concept of authorization through API keys can be implemented in several ways. The most interesting points are how many different keys are used, who manages keys, and what's in a key. To the OLCS client none of this matters, since it knows only its own key, doesn't care where it came from, and it treats it as an opaque token.

Number of keys

- a single key for everyone – this is by far the easiest to set up, but also riskiest; if the key is compromised, revoking it will block all clients. If there are many clients, the risk of compromising the key also increases. Still, this can work nicely for a small number of clients.
- a key per organization/account – this fits a situation where multiple organizations are printing to an external service provider. Every organization must take care of their own key, and if they get it wrong, only their key gets revoked while others remain unaffected. This could also apply to departments in a large organization.
- a key per client – the advantage is that a compromised key has minimal impact. Managing keys could become quite involved, though.

For OL Connect Send deployments by service providers for customers, having at least a key per account is recommended. This also makes it easy to close an account when the customer decides they don't want it anymore. Large accounts could use multiple keys.

Who manages keys

- service provider – users just take the keys that are provided to them. If a user suspects that a key is compromised, they will have to request revocation and a new key. The advantage could be that it may be easier to implement than the alternative below.
- service users – the keys still get created on the server side, but now (admin) users have an interface that allows them to create and revoke API keys for their account as they see fit. Users can take full responsibility for managing keys. This self-service also scales easier.

What's in a key

From the OLCS client's perspective an API key is an opaque token. From the key management perspective and the server side this is not necessarily the case. It's also possible to create a token that has meaningful content that is then encrypted and signed. JSON Web Tokens are an example of this. This opens up interesting possibilities, but also makes verifying an API key more complex. For our sample implementations we have chosen to keep things simple, and just use opaque tokens as API keys.

Protecting API keys

To make authorization through API keys work, they have to be kept private. To achieve this, some care must be taken when handling API keys.

- Client side, administrators should take their usual care and not leave API keys in files or elsewhere after configuration is complete.
- API keys should be provided to users in a secure way. The most obvious way to do this today, is to have them downloaded or copied from a secure website.
- The OLCS client should be configured to only use https for transferring jobs. Which is strongly recommended anyway. Without https, the API key does not add any security at all. This also implies using the Node.js Server Input for receiving jobs in Connect Workflow, because that supports TLS 1.2, and older versions of TLS are no longer regarded as secure.
- When handling job transfers in Workflow, care must be taken to not accidentally leak API keys. Workflow makes it very easy for API keys to accidentally end up in log files, which get shared with others often if an issue occurs. The sample we created takes this into account.

Sample implementation

At the time of writing, the samples mentioned in this article could not yet be published, because no certified version of the OL Connect Send client was available. This means that it would not be possible to deploy a working sample. These samples will be made available later.