Setting up SSL Webhook notifications

Question

Is that possible to get notifications about SSL Order status change?

Answer

Yes, that is possible by setting up a webhooks on customer's end.

Webhook terminology

An event is a certificate change, such as a certificate being requested, issued, or failing validation. Each change has a corresponding Delivery object. Webhook endpoints are URLs defined by users to which Openprovider sends deliveries. A single event may be sent to many webhook endpoints. Webhooks refers to the overall concept of sending notifications to webhook endpoints. Webhooks refers to the overall concept of sending notifications to webhook endpoints.

Use webhooks to be notified about events that happen with certificates in Openprovider. Some events, like pre-validation results or issuance of the certificate, are not the result of a direct API request, so it could be a problem to keep track of the current state of certificates. Webhooks solve these problems by letting you register a URL to which we will send notifications anytime an event happens with a certificate under your account. When an event occurs - for example, when a certificate is issued, Openprovider generates a Delivery object. This object contains all the relevant information about what just happened, including the type of event and the data associated with it. Openprovider then sends the Delivery object to any URLs in your account's webhooks settings via an HTTP POST request. You can find a full list of all event types below.

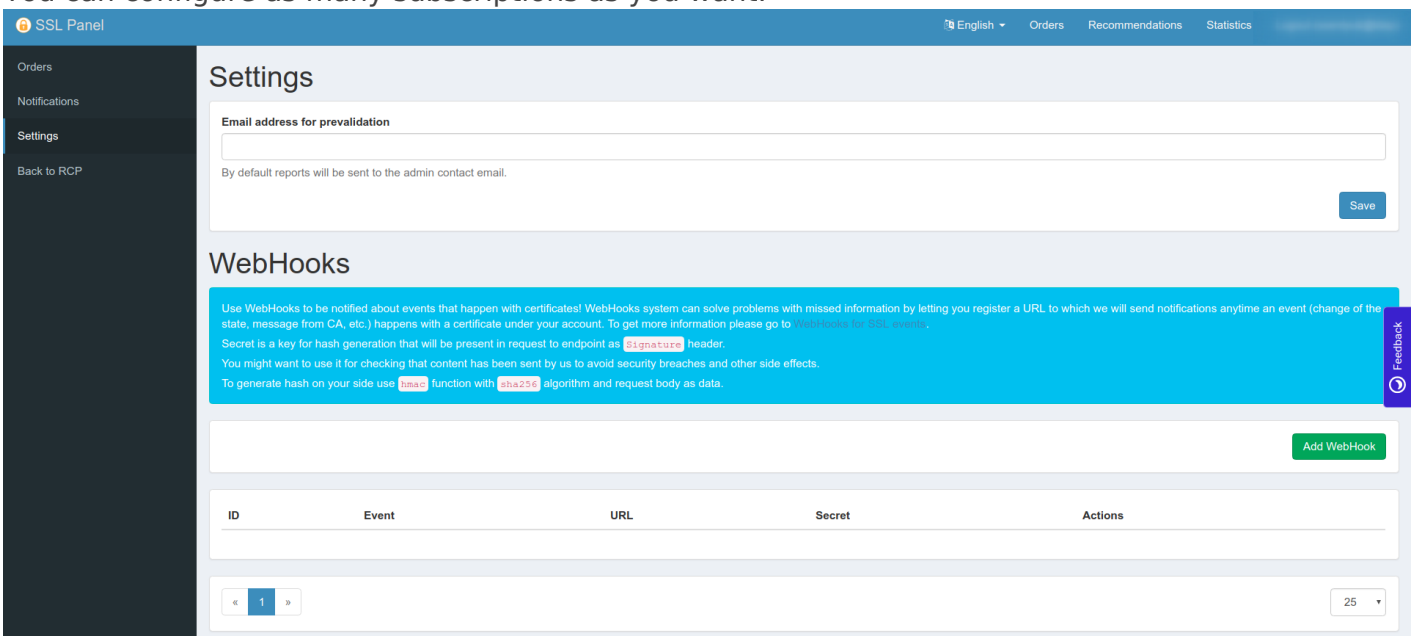
Configuring webhook settings

Webhooks are configured in the Webhook settings section in SSL Panel. Clicking Add endpoint reveals a form to add setting for new subscription.

To configure it you should specify:

- URL where to deliver events
- Event type to be sent
- Secret key that will be used to generate a delivery

You can configure as many subscriptions as you want.



Common webhook mistakes

The two most common mistakes with webhooks are providing the wrong URL in the dashboard or the webhook endpoint not returning a 2xx status code. To test against these cases there is an option to send a test ping to each endpoint. Pings can be sent manually to verify functionality.

Available events:

- ssl_panel.created - Occurs whenever a new certificate is created
- ssl_panel.pre_validation.failed - Occurs whenever a validation fails
- ssl_panel.pre_validation.successful- Occurs when all validations have succeeded
- ssl_panel.requested - Occurs whenever a certificate is requested at the CA
- ssl_panel.ca_chat.new_message_from_ca - Occurs whenever the CA has sent new message to the chat
- ssl_panel.issued - Occurs whenever the CA has issued the certificate
- ssl_panel.will_expire_in_30_days - Occurs 30 days before the certificate expires all events - *
- ssl_panel.reissue_failed - Occurs whenever a reissue fails to be processed
- ssl_panel.renew_failed - Occurs whenever renewal fails to be processed

Data format

A webhook will POST to the URL with all data available in the body.

```
POST /payload HTTP/1.1

Host: localhost:4567
User-Agent: Tribuo/1.0
Content-Type: application/json
Content-Length: 6615
Version: "0.0.1" // Version of delivery structure
Signature: signature // HMAC hex digest of the payload, using the hook's
secret as the key (if configured).
Delivery: b32f3ff1-d7a4-479e-b6ae-1857533caa39 //
UUID of this delivery (for ability to check it later or store it).
Entity-Id: 192356 //
Entity id in application (if specified, for example order id)
Event: ssl.created

{
  // Available metadata
  "metadata": {
    "version": "0.0.1",
    "signature": "signature",
    "delivery": "b32f3ff1-d7a4-479e-b6ae-1857533caa39",
    "event": "ssl.created",
    "entityId": 192356
  },
  "data": {
    // JSON payload from application.
    // Payloads can be completely different from application
to application. That logic must be aligned before usage
of this service.
  }
}
```

Structure: delivery details, status mapping

Rules for status mapping

Openprovider API status	status	certificate.status	certificate.operation
PAI	open	open	none
ACT	open	active	none
open	replaced	none	
FAI	closed	rejected	none
closed	revoked	none	
closed	open	none	
EXP	closed	expired	none
REJ	open	requested	deletecancel
open	active, replaced	deletecancel	

REQ	open	requested	request
open	requested	reissue	
open	requested	renew	

Events:

- ssl_panel.pre_validation.failed
- ssl_panel.pre_validation.successful


```
{
"metadata":
{
    "version":"1.0.0",
    "delivery":"18bb58bc-2731-4d12-adb7-ec1e4fcc0ac7",
    "event":"ssl_panel.created",
    "signature":"sha256=8aca68f56d115c765cf91c1f2d27a7ef7348b698f8eed80aa2a34056adfee690"
},
"data":
{
    "id": "3d8b6599-0794-4c92-b019-ee939395b42a",
     "status": "open",
     "certificate":
     {
        "id": "09a704cb-3598-4324-a56f-eaa792f72a23",
        "status": "open",
        "expireAt": null,
        "operation": "none",
        "commonName": "openprovider.nl",
        "activatedAt": null,
        "certificates":
        {
            "main": null,
            "root": null,
            "intermediate": null
        },
        "preValidationStatus": "failed",
        "preValidationResults":
        [
            {"name": "whois_validator", "status": "passed"},
            {"name": "kvk_validator", "status": "failed"},
            {"name": "phone_number_validator", "status": "passed"}
        ],
        "openproviderId": 123456
    }
}
```


Events:

- ssl_panel.created
- ssl_panel.requested

- ssl_panel.issued ssl_panel.will_expire_in_30_days


```
{
"metadata":
    {
    "version":"1.0.0",
    "delivery":"18bb58bc-2731-4d12-adb7-ec1e4fcc0ac7",
    "event":"ssl_panel.created",
    "signature":
"sha256=8aca68f56d115c765cf91c1f2d27a7ef7348b698f8eed80aa2a34056adfee690"
},
"data":
{
    "id": "38e792f4-90f4-4e37-93ac-0478d01f954c",
    "status": "open",
    "certificate":
    {
        "id": "2410c949-9600-46ac-9c53-62de72742859",
        "status": "open",
        "expireAt": null,
        "operation": "none",
        "commonName": "openprovider.nl",
        "activatedAt": null,
        "certificates":
        {
            "main": null,
            "root": null,
            "intermediate": null
        },
        "preValidationStatus": "passed"
    },
    "openproviderId": 123456
}
}
```


Events:

- ssl_panel.ca_chat.new_message_from_ca


```
{
"metadata":
    {
    "version":"1.0.0",
    "delivery":"18bb58bc-2731-4d12-adb7-ec1e4fcc0ac7",
    "event":"ssl_panel.created",
    "signature":
"sha256=8aca68f56d115c765cf91c1f2d27a7ef7348b698f8eed80aa2a34056adfee690"
},
"data":
{
    "date": "2017-12-08T10:51:40+0100",
    "message": "We try to get contact the organization via the public
```

```
number +111111111 but we do not get anyone on the line. Can you indicate
when we can reached the organization for the validation?",
    "orderId": "0111111-1640-41d4-9b69-71600031d266",
    "openproviderId": "111111"
}
}
```

Receiving a webhook notification

Creating a webhook endpoint on your server is no different from creating any page on your website.
With PHP, you might create a new .php file on your server; with a framework like Laravel, you add a new route with the desired URL.

Webhook data is sent as JSON in the POST request body. The full delivery details are included and can be used directly, after parsing the JSON into an Delivery object.

```
<?php

$body = file_get_contents('php://input'); // get request body
$delivery = json_decode($body, true) // decode delivery as array

// handle event
```

Responding to a webhook

To acknowledge receipt of a webhook, your endpoint should return a 2xx HTTP status code. Any other information returned in the request headers or request body is ignored. All response codes outside this range will indicate to Openprovider that you did not receive the webhook.

If a webhook is not successfully received, Openprovider continues trying to send the webhook once an hour for up to 3 days.

Verifying deliveries

As an extra security measure, you can verify a delivery before acting upon it:

```
<?php

// PHP 7.0+

$body = file_get_contents('php://input');
// get request body$delivery = json_decode($body, true)
// decode delivery as array// get signature that has been generated by sender
[$algorithm, $senderHash] = explode('=', $_SERVER['HTTP_SIGNATURE']);
// generate our signature
$knownHash = hash_hmac($algorithm, $delivery['data'], 'secretPhrase');

// now we can check generated signature against ours
```

```
if (!hash_equals($knownHash, $senderHash)) {
    header('HTTP/1.0 403 Forbidden');

    echo 'You are forbidden!';
}

// handle event
```

Revision #2
Created 2026-08-07 14:53:19 UTC by Sarath
Updated 2026-08-07 16:02:13 UTC by Sarath