

WebHook

- [Structure of the Webhook](#)
- [Validate WebHook](#)
- [Error Management](#)

Structure of the Webhook

The WebHook is sent to the ADDRESS you have stored and contains the following JSON object:

```
{ Guid: '', MessageType: '', DateTime: '' }
```

Guid corresponds to the Guid under which you can retrieve the Data from the API depending on the MessageType.

DateTime is the time at which the Object was created in DeDeSales.

MessageType Definition

Text	Value
Order	0
ReturnOrder	1
NewCustomer	2
CustomerChangeAdress	3

Validate WebHook

The following header is sent with the WebHook:

```
HTTP_X_HUB_SIGNATURE_256
```

If you determine the post data and calculate the hash with your secret, you can ensure that the call actually originated from DeDeSales.

Testing the WebHook Validation

You can use the following `secret` and `payload` values to verify that your implementation is correct:

- `secret`: It's a Secret to Everybody
- `payload`: Hello, World!

If your implementation is correct, the signatures you create should match the following signature values:

- Signature: 757107ea0eb2509fc211221cce984b8a37570b6d7586c22c46f4379c8b043e17
- X-Hub-Signature-256:
sha256=757107ea0eb2509fc211221cce984b8a37570b6d7586c22c46f4379c8b043e17

Examples

You can use a programming language of your choice to implement the HMAC verification in your code. Below are some examples showing how an implementation might look in various programming languages.

Ruby Example

For example, you can define the following `verify_signature` function:

```
def verify_signature(payload_body) signature = 'sha256=' +  
  OpenSSL::HMAC.hexdigest(OpenSSL::Digest.new('sha256'), ENV['SECRET_TOKEN'], payload_body)  
  return halt 500, "Signatures didn't match!" unless Rack::Utils.secure_compare(signature,  
    request.env['HTTP_X_HUB_SIGNATURE_256']) end
```

You can then call the function when you receive a WebHook:

```
post '/payload' do request.body.rewind payload_body = request.body.read
verify_signature(payload_body) push = JSON.parse(payload_body) "I got some JSON:
#{push.inspect}" end
```

Python Example

For example, you can define and call the following `verify_signature` function when you receive a WebHook:

```
import hashlib import hmac def verify_signature(payload_body, secret_token, signature_header):
    """Verify that the payload was sent from GitHub by validating SHA256. Raise and return 403 if
    not authorised. Args: payload_body: original request body to verify (request.body())
    secret_token: GitHub app webhook token (WEBHOOK_SECRET) signature_header: header received from
    GitHub (x-hub-signature-256) """ if not signature_header: raise HTTPException(status_code=403,
    detail="x-hub-signature-256 header is missing!") hash_object =
    hmac.new(secret_token.encode('utf-8'), msg=payload_body, digestmod=hashlib.sha256)
    expected_signature = "sha256=" + hash_object.hexdigest() if not
    hmac.compare_digest(expected_signature, signature_header): raise
    HTTPException(status_code=403, detail="Request signatures didn't match!")
```

JavaScript Example

For example, you can define the following `verifySignature` function and call it in any JavaScript environment when you receive a WebHook:

```
let encoder = new TextEncoder(); async function verifySignature(secret, header, payload) { let
parts = header.split("="); let sigHex = parts[1]; let algorithm = { name: "HMAC", hash: {
name: 'SHA-256' } }; let keyBytes = encoder.encode(secret); let extractable = false; let key =
await crypto.subtle.importKey( "raw", keyBytes, algorithm, extractable, [ "sign", "verify" ],
); let sigBytes = hexToBytes(sigHex); let dataBytes = encoder.encode(payload); let equal =
await crypto.subtle.verify( algorithm.name, key, sigBytes, dataBytes, ); return equal; }
function hexToBytes(hex) { let len = hex.length / 2; let bytes = new Uint8Array(len); let
index = 0; for (let i = 0; i < hex.length; i += 2) { let c = hex.slice(i, i + 2); let b =
parseInt(c, 16); bytes[index] = b; index += 1; } return bytes; }
```

Typescript Example

For example, you can define and call the following `verify_signature` function when you receive a WebHook:

JavaScript

```
import { Webhooks } from "@octokit/webhooks"; const webhooks = new Webhooks({ secret:
process.env.WEBHOOK_SECRET, }); const handleWebhook = async (req, res) => { const signature =
req.headers["x-hub-signature-256"]; const body = await req.text(); if (!(await
webhooks.verify(body, signature))) { res.status(401).send("Unauthorized"); return; } // The
rest of your logic here };
```

C# Example

For example, you can define the following `verify_signature` function:

```
using System; using System.Text; using System.Security.Cryptography; public bool
verify_signature( string header_HTTP_X_HUB_SIGNATURE_256, string postData, string secret ) {
HMACSHA256 mySHA256 = new HMACSHA256(Encoding.ASCII.GetBytes(secret)); byte[] hashValue =
mySHA256.ComputeHash(Encoding.ASCII.GetBytes(postData)); return
header_HTTP_X_HUB_SIGNATURE_256 == PrintByteArray(hashValue); } private String
PrintByteArray(byte[] array) { String result = String.Empty; for (int i = 0; i < array.Length;
i++) result += $"{array[i]:X2}"; return result; }
```

Error Management

If a WebHook cannot be delivered to the URL you specified, the configured WebHook will be automatically deleted if both of the following events occur:

1. 1,000 consecutive failed attempts have been made.
2. The first attempt was more than 24 hours ago.

Upon deletion, you will receive an email notification at the email address you provided when creating the WebHook.