Secure Requests

Most API calls require an access token, but malicious developers can impersonate OAuth Clients or steal access tokens. They will then use these to send spam on the behalf of your app. Instagram has automated systems to detect spam, and will automatically disable the OAuth Clients responsible for these calls. You can mitigate the risk of your app being disabled by restricting some vectors of abuses. This document covers some of the ways you can protect your app.

Disable Client-Side (Implicit) Authentication

The Implicit OAuth Grant flow was created for java-script or mobile clients. Many developers use this flow because of its convenience. Unfortunately, malicious developers can also use this flow to trick people into authorizing your OAuth Client. They can collect access tokens and then make API calls on behalf of your app. When this occurs, your OAuth Client could be banned from the platform by our spam detection systems.

If your app is powered by a server infrastructure, you can disable the Client-Side (Implicit) OAuth flow by checking the **Disable implicit OAuth** setting in your OAuth Client configuration. If checked, Instagram will reject Client-Side (Implicit) authorization requests and only grant Server-Side (Explicit) authorization requests. This setting helps protect your app because the Server-Side (Explicit) OAuth flow requires the use of your Client Secret, which should be unknown to malicious developers.

Important

Your Client Secret should be kept secure at all times. Do not share this Secret with anyone, do not include it in java-script code or a mobile client. Mobile apps that do not have a server-side component should have the **Disable implicit OAuth** setting unchecked. You have the ability to reset your Client Secret to a new value at any time, if you suspect that it was leaked.

Enforce Signed Requests

Access tokens are portable: they can be generated on one machine and re-used elsewhere. Access tokens can also be stolen by malicious software on a person's computer or a man in the middle attack. A stolen access token can then be used to generate spam. When targeted by such abuses, your app could be blocked by our automated systems.

You can secure your API calls and mitigate impersonation attempts by making server-side calls and passing a per-request signature using your Client Secret. Edit your OAuth Client configuration and mark the **Enforce signed requests** checkbox. When enabled, Instagram will check for the *sig* parameter of each request and verify that the value matches a hash computed using your Client Secret. The expected value is a HMAC using the SHA256 hash algorithm with all your request parameters and your Client Secret.

Important

Your Client Secret should be kept secure at all times. Do not share this Secret with anyone, do not include it in java-script code or a mobile client. Mobile apps that do not have a server-side component should not use the **Enforce signed requests** setting. You have the ability to reset your Client Secret to a new value at any time, if you suspect that it was leaked.

Signature format

```
Token to sign: endpoint|key1=value1|key2=value2|...

Parameter name: sig
Parameter value: signed token with your Client Secret using the SHA256 hash algorithm
```

- Token: API endpoint appended with a concatenation of all key/value pairs of your request parameters, **sorted by key** in ascending order. Each key/value pair is separated by the pipe character.

Examples

Endpoint: /users/self Parameters: access_token=fb2e77d.47a0479900504cb3ab4a1f626d174d2d App Secret: 6dc1787668c64c939929c17683d7cb74

With this example, the signature key/value should be:

```
sig=cbf5a1f41db44412506cb6563a3218b50f45a710c7a8a65a3e9b18315bb338bf
```

Endpoint: /media/657988443280050001_25025320 Parameters: access_token=fb2e77d.47a0479900504cb3ab4a1f626d174d2d&count=10 App Secret: 6dc1787668c64c939929c17683d7cb74

Here the signature key/value would then be:

```
sig=260634b241a6cfef5e4644c205fb30246ff637591142781b86e2075faf1b163a
```

The signature describes the hex representation of a RFC 2104-compliant HMAC with the SHA256 hash algorithm, using the API endpoint, your request parameters and your Client Secret. Most programming languages provide the tools to create such a signature. Here are some examples to get you started.

Python

```python
# -*- coding: UTF-8 -*-
import hmac
from hashlib import sha256

def generate_sig(endpoint, params, secret):
    sig = endpoint
    for key in sorted(params.keys()):
        sig += '|%s=%s' % (key, params[key])
    return hmac.new(secret, sig, sha256).hexdigest()

endpoint = '/media/657988443280050001_25025320'
params = {
    'access_token': 'fb2e77d.47a0479900504cb3ab4a1f626d174d2d',
    'count': 10,
}
secret = '6dc1787668c64c939929c17683d7cb74'

sig = generate_sig(endpoint, params, secret)
print sig
```

Ruby

```ruby
require 'openssl'
require 'base64'

def generate_sig(endpoint, params, secret)
  sig = endpoint
  params.sort.map do |key, val|
    sig += '|%s=%s' % [key, val]
  end
  digest = OpenSSL::Digest::Digest.new('sha256')
  return OpenSSL::HMAC.hexdigest(digest, secret, sig)
end

endpoint = '/media/657988443280050001_25025320'
params = {
  'access_token' => 'fb2e77d.47a0479900504cb3ab4a1f626d174d2d',
  'count' => 10,
}
secret = '6dc1787668c64c939929c17683d7cb74'

sig = generate_sig(endpoint, params, secret)
print sig
```

PHP

```php
<?php
function generate_sig($endpoint, $params, $secret) {
  $sig = $endpoint;
  ksort($params);
  foreach ($params as $key => $val) {
    $sig .= "|$key=$val";
  }
  return hash_hmac('sha256', $sig, $secret, false);
}

$endpoint = '/media/657988443280050001_25025320';
$params = array(
  'access_token' => 'fb2e77d.47a0479900504cb3ab4a1f626d174d2d',
  'count' => 10,
);
$secret = '6dc1787668c64c939929c17683d7cb74';

$sig = generate_sig($endpoint, $params, $secret);
echo $sig;
```

Testing Signed Requests

Using an invalid signature will cause your API calls to fail if the **Enforce signed requests** setting is set. Because of this, you may want to test this parameter before you enable the setting for production code.

Fortunately you can use cURL to test your header format and signature easily:

```
curl \
  -X POST \
  -F 'access_token=<your_access_token>' \
  -F 'sig=<your_signature>' \
  https://api.instagram.com/v1/media/657988443280050001_25025320/likes
```

Common responses:

REASON	RESPONSE
Success	{"meta":{"code":200},"data":null}
Signature is required	{"code": 403, "error_type": "OAuthForbiddenException", "error_message": "Missing required parameter 'sig'"}
Failed to validate signature	{"code": 403, "error_type": "OAuthForbiddenException", "error_message": "Signature does not match"}