Phishing with OAuth and o365/Azure

By Etienne Stalmans

Published: 2017-08-02 · Archived: 2026-04-05 18:17:37 UTC

Typically phishing has provided a low tech approach to getting access to credentials and services. The mainfocus up until now has been on getting username&passwords or tricking users into executing code. Subsequently, user awareness has gone up and users are better at identifying suspicious pages. Experience has shown that the click-through rate on emails have remained high, while users have been (slightly) less likely to enter credentials and more likely to report the phishing page. Recently, while digging into the authentication options available in Office365, I realised that OAuth could allow for a stealthy phishing mechanism. The benefits that OAuth provide are numerous;

1. Not your average phishing page / something new
2. Hosted by our target (o365) - so no dodgy domains or trikery
3. If users aren't entering actual credentials, they are less likely to report that they made a mistake

Below I'll outline the process for crafting a phishing 'app' and using this with Office365.

Disclaimer: I have reported this to MSRC and was told it appears to be a UI/Design issue and not a security issue.

What is OAuth

OAuth is designed to allow for third-party application access to Microsoft services, without disclosing user credentials, or constantly asking users to reauthenticate. Users log in once, give an application permission to access certain functions (scopes) and then forget about it.

The design and implementation of OAuth in Office365/Azure is pretty well laid out in the MS Documentation.

If you follow the link, you'll find a good description of the OAuth flow, namely:

1. User navigates to: https://login.microsoft.com/common/oauth2/v2.0/authorize
2. User grants permission to your app and a redirect occurs to: https://your.application.host/authrized
3. Your app requests an OAuth token from MS and then redirects the user, either to MS or any arbitrary site.

The only *link* that your phishing target sees in this case is the https://login.microsoft.com/common/oauth2/v2.0/authorize link. It meets all the criteria users are trained to check for. It goes to the Microsoft domain, it has HTTPS, it loads Microsoft content and if the user is logged in, will actually greet them with their correct account details.

Building the Phish

For this you'll require a valid Microsoft 'live' account, I'm simply using an Outlook account.

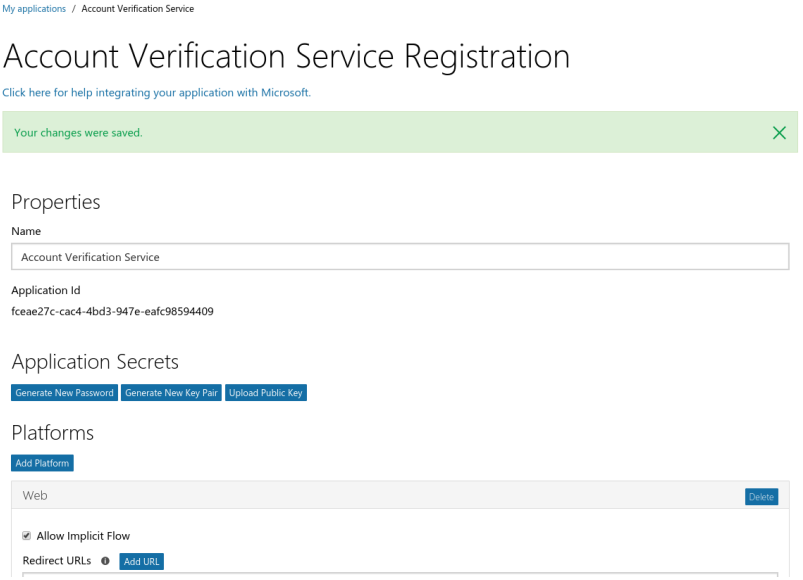
The "application"

Firstly you'll need to create an *app* at apps.dev.microsoft.com. Here you'll simply be registering a valid application with Microsoft, giving it a name and defining the call back URLs.

It's really easy, just follow the on-screen wizard!

1. Set the application name (use something convincing)
2. And set your Redirect URL

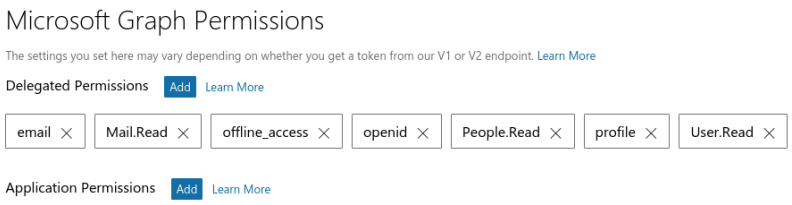
The *Application Name* is what will be displayed to the user, and this is what needs to convince them to trust your app or not. I've gone with **Account Verification Service**, the reasons for this will become clear soon. Oh, it also provides you with a convincing reason for users to click through, ie: "Your account needs to be verified…", "Your account has expired…", "Confirm your account details…" etc etc.



Creating your App

Next, you will need to use *add platform* to select *web*. Insert a *Redirect URL* – In version 1.0 of the OAuth API (and for non-o365 accounts) Microsoft will display the hostname to the user, just below the application name. Choose wisely if you are planning on using v1.0. Microsoft also forces HTTPS here, so ensure that you have a valid certificate on your site, otherwise the browser is going to be throwing up nasty warnings and the game will be up.

Finally you need to select the permissions you want a user to grant your application. The permissions you choose are up to you, but try avoid selecting all permissions, this could result in raising suspicions. A useful permission to have is *offline_access*, as it allows is for your application to reauthenticate, without user credentials or user interaction, even once the access token has expired. Typically your app is only issued with an authorisation token, which is only valid for set period of time. When this expires, you'll need to get a new token, but the user will have to authorise your app to do this. With the offline_access permission, your app is issued with a *refresh token* token, which allows reauthentication at any point in time, even if your actual access token has expired.



Select your permissions

The "redirect"

The last thing to setup is the actual *redirect url* application; this is where the user is redirected to, once they authorise your application, and you can get the code you need to request an access token from Microsoft.

For this I've written a bit of Go (sorry, I thought about re-coding this into Python and then got over it..)

Redirect Application

There are a few parts to change, namely *client_id*, *scope* and *redirect_url* (I've commented these in the code). The application will be served over HTTPS and uses your *cert.pem* and *cert.key*. These should be valid for your domain and it's up to you where you get them.

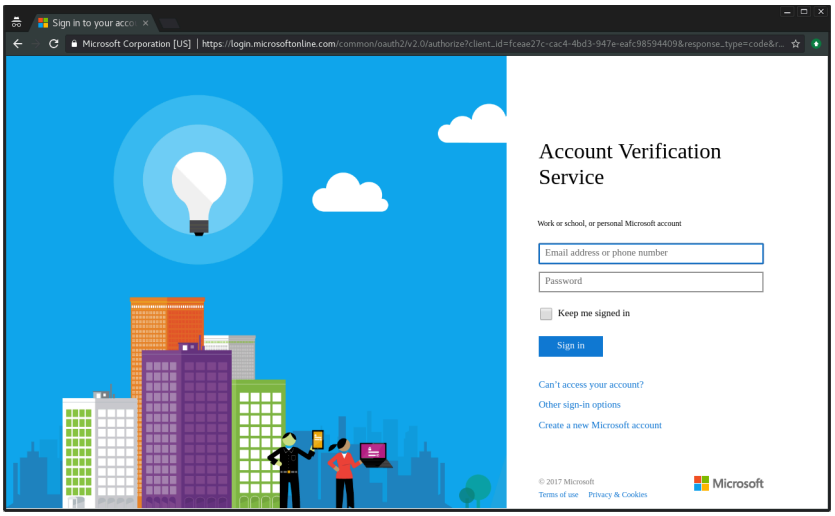
Phishy Phish

You'll now be able to send your phishing email with the link to authorise our application.

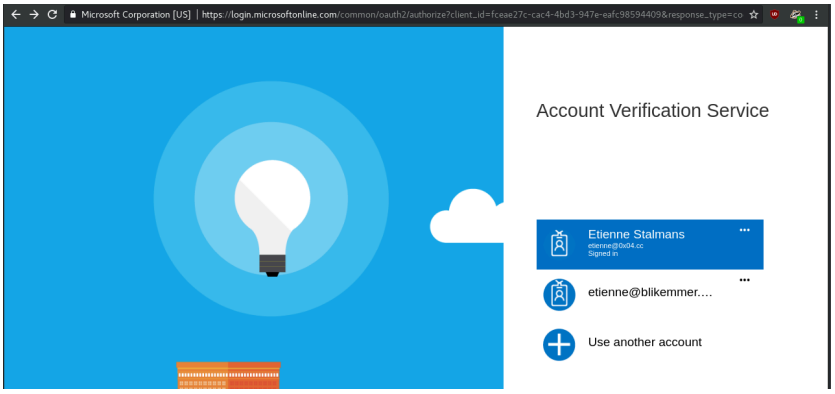
The link should be changed to match your *client_id* (application id), your *redirect_url* and *scope*. If you wish you can also change the *state* but remember that you'll need to change the *state* in the Go script as well (this is used as a CSRF token).

```
https://login.microsoftonline.com/common/oauth2/v2.0/authorize?client_id=fceae27c-cac4-4bd3-947e-eafc98594409&response_type=code&redirect_uri=https%3A%2F%2Fyour.application.host%2Fpermission&response_mode=query&scope=openid%20offline_
```

Sending this link out, hopefully a user takes the bait and clicks. Once they do this, they should see one of the following:

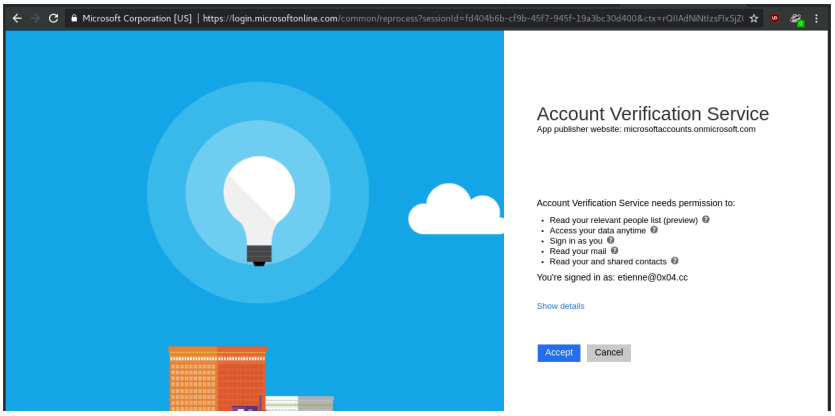


If they haven't logged in yet



If they are logged in

The user will either need to login or confirm the account they wish to use. They will then be redirected to the permission page. If using *v1.0* of the API, the user will be shown the domain used in your *redirect_url* - fortunately for us, v1.0 doesn't actually work that well with o365 anymore, and we can default to *v2.0*. Version 2 has a serious design flaw; your *App publisher website* is shown as **microsoftaccounts.onmicrosoft.com**, winning!



The permissions page

Once the user clicks *accept* they will be redirected to our *redirect_url*, where the Go script will extract the authorisation code, request an OAuth token from Microsoft and redirect the user to *https://outlook.office365.com*. All of this happens without warning and the user would only "see" a redirect to Outlook.

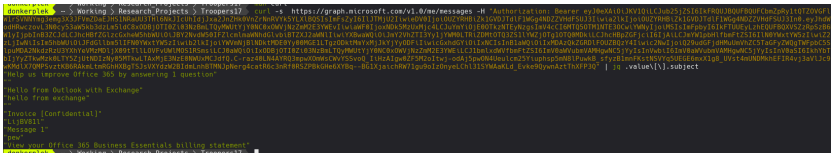
Requesting the OAuth token

This token retrieved includes our *refresh_token* so save this somewhere for future use.

Exploit

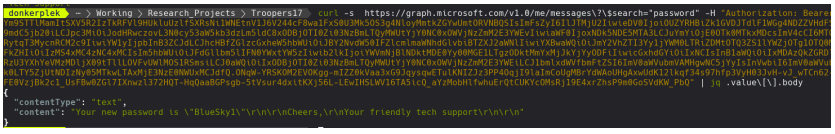
Now that you have a valid OAuth token, this can be used to interact with the target user's account. A detailed list of API calls is available through MSDN.

A few quick examples, how about retrieving the messages and listing their subjects? Use:
https://outlook.office.com/api/v2.0/me/messages



Request a short list of emails & subjects

The API also allows for search, so you could search for juicy info, such as passwords. Use:
https://outlook.office.com/api/v2.0/me/messages?search="password"



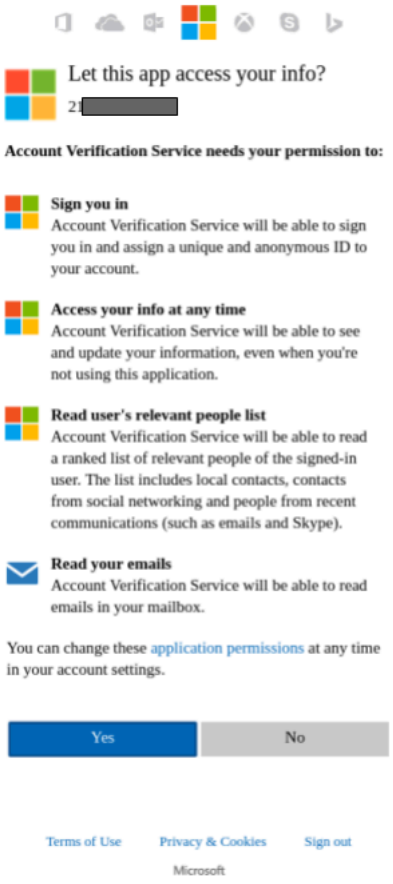
Show emails containing the word *password*

Fin

There are a bunch of attack options available here, with the graph API providing a ton of interesting functionality. How you choose to use this is completely up to you. I find it to be a nice means for gaining a foothold, you could even start sending mail using the compromised account (if you requested the correct permission set).

Fix

This is very similar to the OAuth issue that affected Google Apps not too long ago. That incident, as mentioned in the linked article, was particularly successful as it presented users with a valid interface, and no indication that something was amiss. The same steps should be taken here to make it clear to users that this is a third-party app requesting access. I guess this is probably worm-able as well. For the curious, "Live" accounts get a slightly better dialog. At least it displays the "app publisher webpage" (in this case an IP address)



Live dialog

Source: https://staaldraad.github.io/2017/08/02/o356-phishing-with-oauth/