Microsoft 365 OAuth Device Code Flow and Phishing

By By: Daniel Min

Published: 2021-08-17 · Archived: 2026-04-05 23:34:17 UTC

During a recent red team engagement, we found that the target organization used a well-known identity access management (IAM) product for their multi-factor authentication (MFA) solution. Most of their Internet-facing login portals, such as Office365 and Citrix portals, were behind this IAM platform; however, their users were allowed to enable multiple MFA options, including Email OTP (one-time passcode). Given these situations, we thought a phishing campaign abusing Microsoft 365 OAuth device code flow would be a great attack vector for potentially bypassing their MFA and gaining access to the applications from the Internet.

Figure 1: Email OTP Enabled for MFA

Microsoft OAuth Authentication Flow

According to Microsoft, it uses OAuth device authorization grant to allow users to sign in to input-constrained devices such as smart TVs, IoT devices or printers. To enable this authorization flow, the device will have the user visit a webpage from another device's browser to sign in. Once the user signs in, the device can request access tokens as needed.

To explain what this is doing:

1. A user initiates an application on a device, which supports this device authorization grant flow.
2. The application connects the **/devicecode** endpoint with the **client_id** of the application and the permission scopes, such as **Files.ReadWrite**.
3. The **/devicecode** endpoint sends back **user_code, device_code** and **verification_uri**.
4. The user now visits the **verification_uri** and submits the **user_code**.
5. Azure will prompt the user to complete the authentication process and accept the presented **permission scopes**.
6. The device continuously connects to the **/token** endpoint with **client_id** and **device_code**. After the user's successful login, it gets **access_token, refresh_token** (if **offline_access** scope was consent) and **id_token**.

How Can We Abuse This for Phishing?

1. An attacker creates a fake Azure App leveraging Azure Active Directory.
2. The attacker requests **user_code, device_code** and **verification_uri** by connecting to **/devicecode** endpoint with **client_id** of the created Azure App.
3. After receiving necessary items, the attacker sends a phishing email to a victim soliciting them to visit the **verification_uri** and enter the **user_code**.
4. The victim submits the **user_code** and completes the normal Microsoft sign-in, and accepts the permission consent.
5. The attacker requests **access_token** and **refresh_token** from **/token** endpoint to impersonate the victim.
    a. The **access_token** can then be used to access the victim's Office365 products, including Outlook mail leveraging Microsoft Graph API.
    b. By default, the **access_tokens** are valid for 60 days and **refresh_tokens** are valid for a year Thus, with **refresh_token**, one can continuously re-request for the victim's **access_token** for persistence purposes.

Practical Example of Microsoft OAuth Device Code Phishing

One perk about this phishing technique is that we do not need to build custom phishing infrastructure (e.g., creating a phishing server and landing page). We can use pretty much everything provided by Microsoft since we are technically

leveraging their legitimate OAuth authentication flow.

Creating Azure Application

First, we need to create an Azure application. This application will serve as an endpoint where we will generate **client_id** and **device_code**, which are later used for obtaining the victim's **access_token, refresh_token** and **id_token**.

Steps: Azure Portal → Azure Active Directory → App registrations → New registration

Figure 3: Azure App Registrations

Next, enter a name for the application. This can be anything such as "Optiv Remote User Management", etc. For **Supported account types**, select "Accounts in any organizational directory (Any Azure AD directory – Multitenant) and personal Microsoft accounts (e.g., Skype, Xbox)" so that target users could use their business Microsoft accounts to allow the application consents.

Figure 4: Azure Register an Application

This will create our Azure application and make a note for the **Application (client) ID** value. This will serve as **client_id** value.

Figure 5: Azure App client_id

Finally, change **Allow public client flows** to "Yes" under the **Authentication** section and select "Save."

Figure 6: Azure App Authentication

Launching Phishing Campaign

To conduct the device code phishing, we first need to make a POST request to **/devicecode** endpoint to obtain **user_code** and **device_code**. Postman, an API testing and development platform, can be very useful for this.

1) HTTP POST Request to /devicecode	
Endpoint	https://login.microsoftonline.com/organizations/oauth2/v2.0/devicecode
client_id	54c9f794-489c-4473-8407-XXXXX (**Application ID from the Azure App*)
scope	Contacts.Read Files.ReadWrite Mail.Read Notes.Read Mail.ReadWrite openid profile User.Read email offline_access

Note: A full list of the **scope can be found at **Microsoft.com**. Make sure to carefully check permissions required for each **scope** since some require admin consent.*

Figure 7: POST Request to /devicecode Endpoint

Next, upon obtaining the **user_code**, we need to send a phishing email to the victim. This is the only tricky part of this phishing attack since you need to lure the victim to visit the **verification_uri** and enter the **user_code**. Another caveat is that obtained **user_code** and **device_code** will expire 15 minutes after they have been generated. Thus, if they expire, you will be required to regenerate them and resend the phishing email.

For our engagement, we came up with the following phishing email template:

Figure 8: Example Phishing Email

We registered our phishing domain to send emails from Microsoft Outlook. Also, the reason why we did not embed the hyperlink for the **verification_uri** (https://microsoft.com/devicelogin) was that our phishing email kept landing in the spam folder. Though we are not 100% sure, we think Microsoft may have done some subtle filtering for hyperlinked **verification_uri** and **user_code** combinations not originating from a Microsoft email domain. We observed this behavior from our dynamic testing at that time, so we decided to make our phishing email like the above.

If the phishing attempt was successful and the victim visited the **verification_uri**, entered the **user_code**, and completed authentication, the victim would have seen the following devicelogin and permission consent pages:

Figure 9: Microsoft Devicelogin Page for Entering user_code

Figure 10: Example Sign in Page

Figure 11: Example Device Permissions Requested Page

Figure 12: Example Final Page After Accepting Permissions

As for the attacker, they will need to keep checking whether the victim completed the authorization process by sending a POST request to /token API endpoint after the phishing email is sent. Once the authorization process is completed, it will present a bearer token formatted **access_token** and **refresh_token**, which you can use to mimic the victim's access.

2) HTTP POST Request to /token	
Endpoint	https://login.microsoftonline.com/organizations/oauth2/v2.0/token
client_id	54c9f794-489c-4473-8407-XXXXX (**Application ID from the Azure App*)
code	<device_code> (**Example: FAQABAAEAAAD--DLA3VO7QrddgJg7WevrafD5YlswpVDFrAq_avAeRuloI07HQehfaP-8QEeTzrP0k24tIrsNKgrltMt7RIsWieb_OJhqJInked8PpsCh5CMhwFqakM0y2sG3fphtxTevtxozvVk5rR_xOCKPmx W5WkvHGOjuQwlAOFB7Qo4ni8PjfAYRKx6tGqv39c9_sIgAA*)
grant_type	urn:ietf:params:oauth:grant-type:device_code

Figure 13: Authorization Pending

Figure 14: Authorization Granted & Gain Bearer access_token

For example, with the **access_token**, the attacker can now access the victim's email inbox using Microsoft Graph API.

Figure 15: Accessing Victim's Outlook Mail Inbox

Automation of Device Code Phishing

Going back to our red team scenario, our goals were to:

1. Launch a Microsoft 365 device code phishing attack
2. Gain access to the target user's inbox (**The targets were ones whom we had already compromised passwords for via password spraying against their MDM solution. We used airCross created by Matt Burch to perform a password guessing attack.*)
3. Trigger the Email OTP for MFA options
4. Read the created Email OTP code from the victim's inbox
5. Bypass MFA

We originally found a great open-source framework, TokenTactics created by revrsh3ll to conduct our phishing attack. The tool was designed to contain all the necessary functions and tactics to abuse Azure JSON Web Token (JWT). We highly recommend trying it out. Leveraging his tool, however, we decided to create a simple Python script to automate some of the device code phishing flow to solely meet our goals. The first function **getDeviceCode()** was created to request for **user_code** and **device_code**.

Figure 16: getDeviceCode

Figure 17: Example of Running devicePhish.py – Getting user_code & device_code

Next, using the newly obtained **device_code**, the **getAccessToken()** function continuously loops through the authorization process. If the phishing attack is successful, it will yield an **access_token** and **refresh_token**; otherwise, the loop will end after 15 minutes due to the token expiration.

Figure 18: getAccessToken

Figure 19: Example of Victim Entering user_code

Figure 20: Example of Running devicePhish.py – Obtaining access_token & refresh_token

With **access_token**, the **getMail()** function keeps looking for an email containing specific text. For our case, it was looking for a string "MFA – One Time Code" within the email text, so if we were to trigger the Email OTP, it would have sent the temporary code to the victim's inbox. If it finds the right email, it prints out the email context with the OTP code and saves the **email_id** value.

Figure 21: getMail

Figure 22: Example OTP Email

Figure 23: Example of Running devicePhish.py – Reading OTP Email

Finally, with the stored **email_id** value, the **deleteMail()** function will delete the OTP email from the victim's inbox.

Figure 24: deleteMail

Figure 25: Example of Running devicePhish.py – Deleting OTP Email

The script used in this demo can be found here: Microsoft365-devicePhish

Mitigation of Microsoft OAuth Device Code Phishing

This device code phishing attack can be mitigated by making configuration changes for Azure AD. Under the **Enterprise applications** settings, prevent users from giving consent to all applications.

Figure 26: Mitigation

If the above solution is too restrictive, then please check out the following best practices provided by Microsoft:

- Configure how end-users consent to applications
- Managing consent to applications and evaluating consent requests

References

- https://o365blog.com/post/phishing/
- https://threatpost.com/microsoft-seizes-domains-office-365-phishing-scam/157261/

- https://github.com/rvrsh3ll/TokenTactics
