OAuth’s Device Code Flow Abused in Phishing Attacks

secureworks.com/blog/oauths-device-code-flow-abused-in-phishing-attacks

Secureworks Adversary Group and Counter Threat Unit Research Team

In business email compromise (BEC) scams, threat actors can bypass multi-factor authentication (MFA) by abusing OAuth applications. These “illicit consent” attacks typically involve the threat actors using phishing to convince a victim to grant sensitive permissions (e.g., the ability to read and write email) to an attacker-controlled Azure application. One variation on this attack leverages the OAuth interactive device authorization protocol to

imitate legitimate (and possibly verified) OAuth applications during the user consent experience. This approach does not require the threat actor to register a malicious OAuth application, making it challenging for defenders to detect and audit.

Abusing OAuth2.0 Device Authorization

Reports of high-profile BEC incidents and takedowns of threat actors conducting sophisticated phishing campaigns that used COVID-19 (also known as coronavirus) themes highlight the need to detect and prevent OAuth-based threats. In parallel with other research teams, Secureworks® researchers identified a novel phishing technique that abuses the OAuth2.0 Device Authorization Grant protocol (RFC 8628) to obtain illicit consent.

The OAuth 2.0 Device Authorization Grant allows a user to authenticate an application to an OAuth provider from a different device. This authentication flow was created to facilitate authorization of a device that has limited interactivity (e.g., no keyboard). One of the most common implementations is using a mobile phone or PC to authorize a video-streaming application on a device like a Roku. The application on the device provides a code to the user and then polls the provider to detect when authentication happens (see Figure 1).

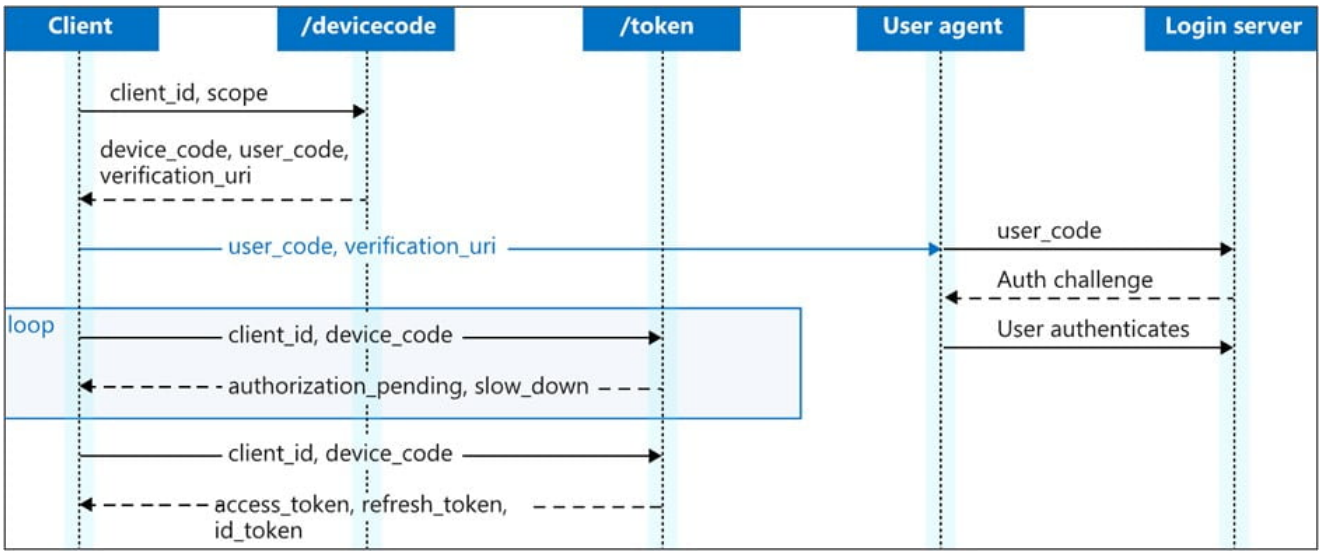

Figure 1. Microsoft Azure Active Directory OAuth 2.0 device flow. (Source: Microsoft)

While investigating the device code flow, Secureworks researchers observed implementations that required only a client ID to generate the user code and device code and then subsequently poll for authentication. This behavior seemed to indicate that a threat actor could use the flow to conduct a phishing attack and impersonate legitimate client applications. The RFC notes this possibility and provides some mitigations that the OAuth provider should have in place (see Figure 2). Impersonating legitimate client applications helps the threat actor avoid detection and increases the likelihood that a victim will click a link.

```
5.4.  Remote Phishing

   It is possible for the device flow to be initiated on a device in an
   attacker's possession. For example, an attacker might send an email
   instructing the target user to visit the verification URL and enter
   the user code. To mitigate such an attack, it is RECOMMENDED to
   inform the user that they are authorizing a device during the user-
   interaction step (see Section 3.3) and to confirm that the device is
   in their possession. The authorization server SHOULD display
   information about the device so that the user could notice if a
   software client was attempting to impersonate a hardware device.
```

Figure 2. Reference to phishing potential and mitigations in RFC 8628. (Source: Internet Engineering Task Force (IETF)).

From a victim's perspective, the authentication process could appear legitimate. They could be prompted to visit a legitimate OAuth provider link and enter the code from a phishing email or SMS message to authorize their phone with an application their company uses (see Figure 3).

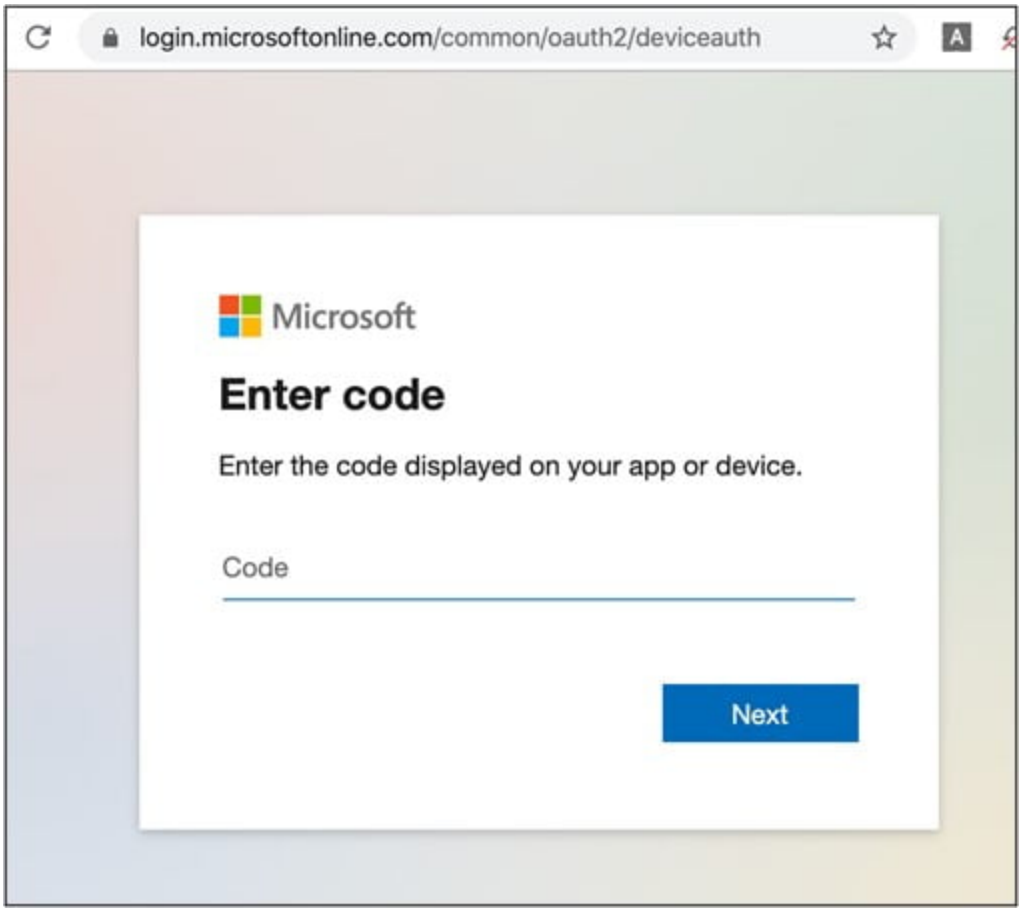

Figure 3. Microsoft device login page. (Source: Secureworks)

After authenticating, the victim could grant permissions to an app that does not appear suspicious. In some cases, the app may be verified (see Figure 4). The permissions could allow the threat actor to perform various tasks, including reading email. If offline access is

specified, the attacker could maintain access long term.

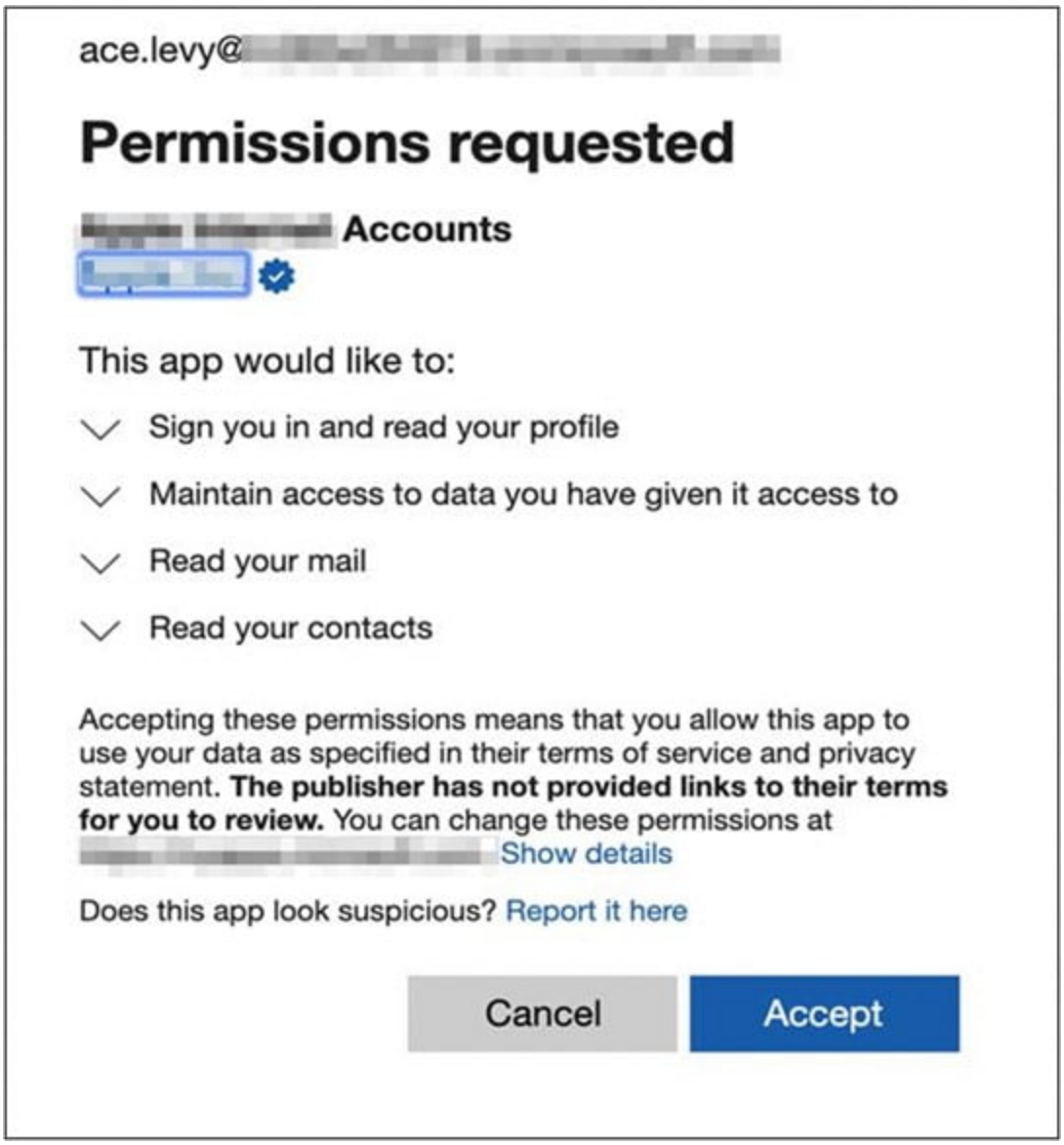

Figure 4. Phishing attack using the OAuth device code flow and a verified application. (Source: Secureworks)

PhishInSuits tool

The Secureworks Adversary Group (SwAG) and Counter Threat Unit™ (CTU) research team developed the PhishInSuits (pis.py) tool to conduct security assessments and test control frameworks against scenarios such as BEC attacks. It combines this variation of illicit consent attack with SMS-based phishing to emulate BEC campaigns and includes automated data-exfiltration capabilities.

PhishInSuits automates the OAuth 2.0 Device Authorization Grant processes, obtaining a user's access/refresh tokens upon application authorization and performing data exfiltration using the acquired access token(s). The tool performs four steps (see Figure 5):

1. Device code authorization: It uses a specified application client ID to communicate with the authorization server and acquire both a user code and a device code.
2. Phishing/smishing: It uses the acquired user code to perform one of two actions:
    1. It sends the target victim(s) an SMS text message containing both the user code and authorization URL and prompts the victim to enter the code and authorize the application.
    2. It instructs the tool's user to send the user code and authorization URL to the victim via email.
3. Polling for authentication: PhishInSuits polls the authorization server for the victim's access and refresh tokens. Polling requests continue at a specified interval until the victim authenticates (which generates an access token) or the device code expires. The tool then writes the full token response from the authorization server to a local file.
4. Data exfiltration: If the tool's user instructs PhishInSuits to exfiltrate data, then the tool uses the acquired access token to query the target API for the specified data. Each API response is written to a local file.

```
$ python3 pis.py -e user@example.com -p '<victim_phone>' \
                 -P'<from_phone>' -s '<twilio_sid>' \
                 -t '<twilio_token>' -c '<client_id>' --get-data


[INFO] [user@example.com] Code successfully retrieved.
[INFO] [user@example.com] Message: To sign in, use a web browser to open
the page https://microsoft.com/devicelogin and enter the code E2XZV7VLV to
authenticate.
[INFO] [user@example.com] Text message successfully sent.
[INFO] [user@example.com] Polling for user authentication...
[INFO] [user@example.com] Polling for user authentication...
[INFO] [user@example.com] Polling for user authentication...
[INFO] [user@example.com] Polling for user authentication...
[INFO] [user@example.com] Polling for user authentication...
[INFO] [user@example.com] Token info saved to user@example.com.tokeninfo.json
[INFO] [user@example.com] Azure Graph API results for 'profile' saved to
user@example.com.profile.json
```

Figure 5. Example output of the PhishInSuits tool. Twilio is a third-party API for programmatically sending SMS messages. (Source: Secureworks)

The tool user could use external (e.g., LinkedIn) or internal (e.g., the organization's Global Address List (GAL)) sources to perform reconnaissance and identify email addresses and phone numbers used by target victims. After compiling a list of victims, the tool user can leverage PhishInSuits to perform targeted phishing via the command shown in Figure 6.

```
$ python3 pis.py -f targets.csv -P '<from_phone>' \
                 -s '<twilio_sid>' -t '<twilio_token>' \
                 -c '<client_id>' --get-data
```

Figure 6. Example command to perform an attack against multiple victims. (Source: Secureworks).

If a victim authorizes the application sent by the tool user, PhishInSuits uses the '--get-data' flag to obtain sensitive data from the Azure Graph API using the victim's access token. The tool user can leverage the victim's access token and the '--api' flag to collect user data from one or more API endpoints.

Mitigations for these attacks

Microsoft offers multiple user consent settings for Azure AD tenants. These configurations include preventing users from granting consent to unapproved applications without administrator preapproval, allowing users to request consent from administrators, restricting the types of consent that can be granted (e.g., disabling permissions to read user mailboxes), and restricting consent to only apps from verified publishers.

Microsoft published best practices for granting consent and managing application permissions. Additional policies and controls for OAuth applications are available in the Microsoft Cloud App Security (MCAS) portal.

Conclusion

BEC continues to be a lucrative type of attack. Combining weaknesses in an OAuth implementation with phishing can improve threat actors' chances of success. By demonstrating these techniques, the PhishInSuits tool can help organizations simulate these attacks to educate employees and identify potential detection and defense methods.

Try the Secureworks PhishInSuits tool.