How to maintain persistent access in a SaaS-native company

By Luke Jennings

Archived: 2026-04-05 14:53:38 UTC

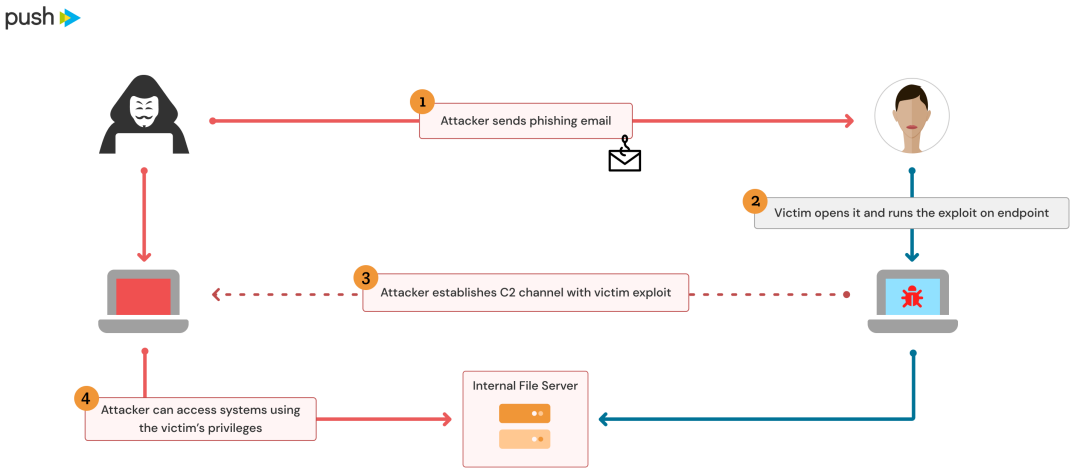
As an attacker, we have a wide range of persistence options available to us in a traditional account or endpoint compromise scenario. From discovering a user's password, to creating new backdoor accounts, to using one of an insane number of "run keys" to keep an implant running beyond reboot, or even moving laterally to other internal systems - an attacker has plenty of choice.

But how does this change in a SaaS-first world? In this post, we'll consider some of the new challenges and opportunities that are presented to an attacker who wants to maintain persistence in the new world order, so you can better investigate incidents and quickly defend against attacks. We'll cover a variety of techniques, including malicious mail rules, OAuth backdoor tricks and document sharing links to see how persistence can be maintained, even in the event of password changes and device wipes.

So what's changed?

In a traditional compromise scenario, a common example would be an endpoint compromised through phishing, which is used to deliver a malicious implant to establish a command and control channel with the endpoint. In order to maintain access, an attacker would likely use one or more endpoint persistence methods to ensure their implant is launched again post-reboot when the user turns their laptop off for the day.

This would often become a foothold into the internal network of the compromised organization. The endpoint or user is the start, but an attacker may seek to move laterally to other endpoints and servers on the internal network, where security is often much lower than the external perimeter.



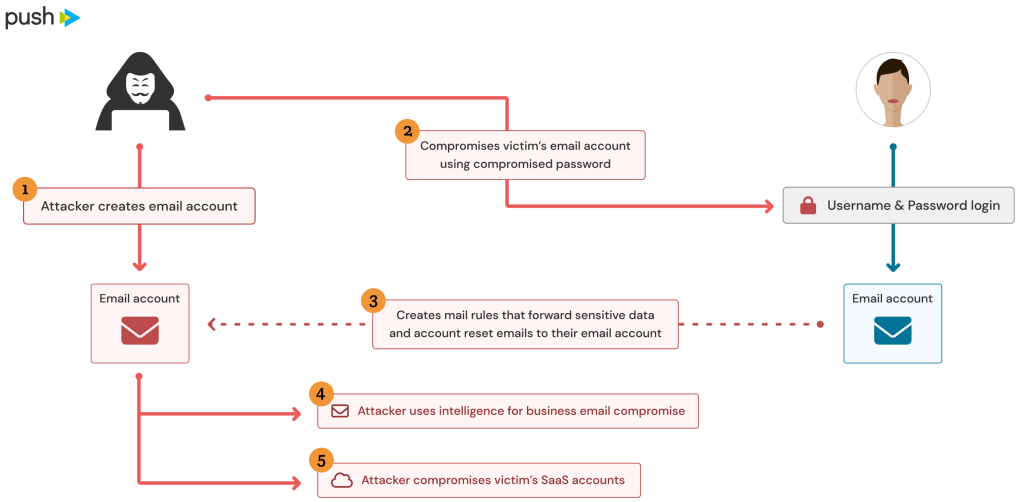
The old way: an attacker gains persistent access through the user's endpoint

In a SaaS-first world, this situation has begun to change somewhat. There are many companies now that have significantly reduced the size of their internal networks or are even fully in the cloud and do not have any internal network infrastructure. In this case, traditional lateral movement becomes much more difficult or impossible. Additionally, endpoints are becoming increasingly hard targets to compromise and incident response teams have matured and have gotten better at cleaning up endpoint compromises.

The consequence of this is that attackers need to make the most use of the access they have during an endpoint or user compromise and maintain access where possible, even in the event of a password reset and full laptop wipe. Additionally, new SaaS-focused persistence options are now possible, which are also often resistant to password changes and endpoints wipes, so these are increasingly attractive options for an attacker.

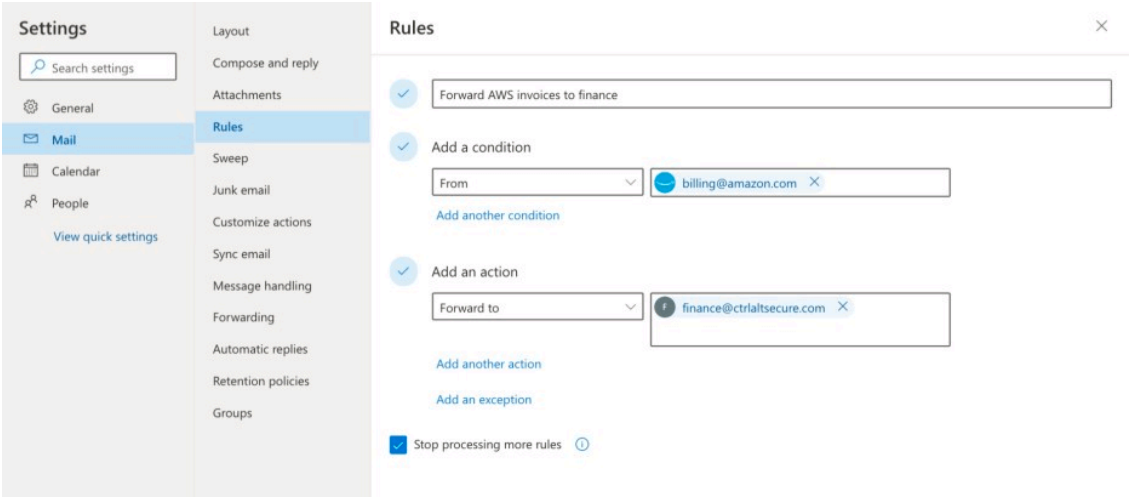
One other change is that persistence is less binary than it has been traditionally. Typically, persistence would often be on a per-user or per-endpoint basis. Either an attacker would have full control of a user account (e.g. knowing the password) or full control of an endpoint (e.g. an implant running on the endpoint). The main differentiation would be in whether endpoint-level access was administrative level control over the endpoint or an implant running as a low-privileged user account. However, in the SaaS-world persistence is much more asset dependent and thus less binary. It could be persistent access to email, or documents, or chat conversations or any number of other assets and capabilities.

Mail rules



An attacker abusing mail rules to gain persistent access

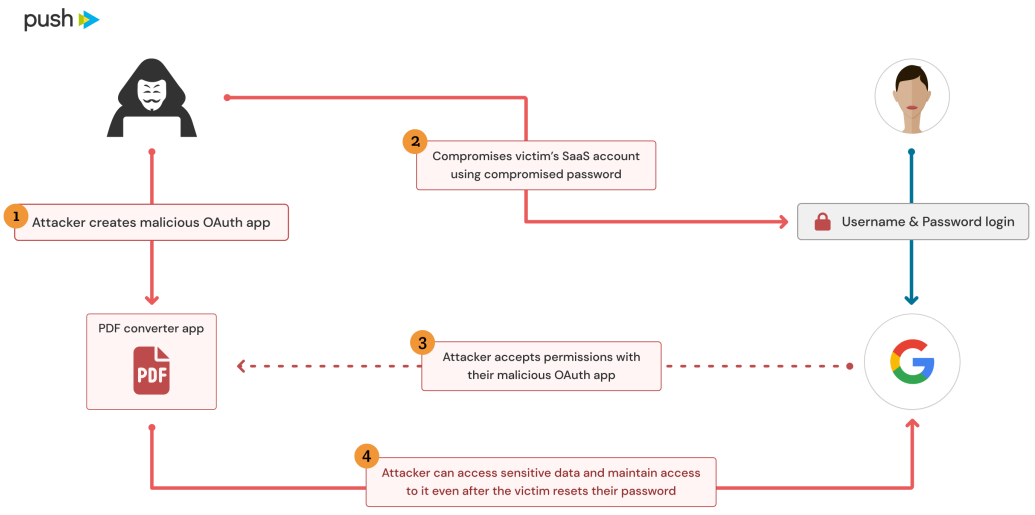
Mail rules are a handy feature found in most email clients. You might have used them to forward emails to your teammates while you're off sipping Piña Coladas, or to move incoming email from that spammy colleague to the "don't read" folder. However, they can also be used for a range of malicious activities, such as forwarding emails to an external address (e.g. password resets, invoices, "confidential" emails etc) or deleting emails (e.g. security alerts!). A good example of a real-world attack involving this technique was the 2020 SANS breach.



Example of a malicious mail rule redirect

If you want to read more about this technique, you can check out our previous article.

OAuth attack #1: Custom OAuth app integration



Attacker uses OAuth integrations to gain persistent access

OAuth apps can be used to request permanent access to a set of permissions on behalf of a user. This can be as simple as the ability to verify a user's identity for a simple social login or it could be as permissive as having full control over email, document stores, wiki pages, admin capabilities, etc. You can read more details about this in our previous article.

However, from an attacker's perspective a custom OAuth app could be created with sensitive permissions and connected to a user's account in order to maintain access to their data. In the event that an attacker has compromised a user's account or endpoint, they could directly consent to their own malicious OAuth app on behalf of the user in order to gain persistence. This could also be achieved as part of a consent phishing attack to effectively compromise a user's account and gain this persistence at the same time. In either case, this would enable continued access to the user's data even if their password is changed and their endpoint fully wiped.

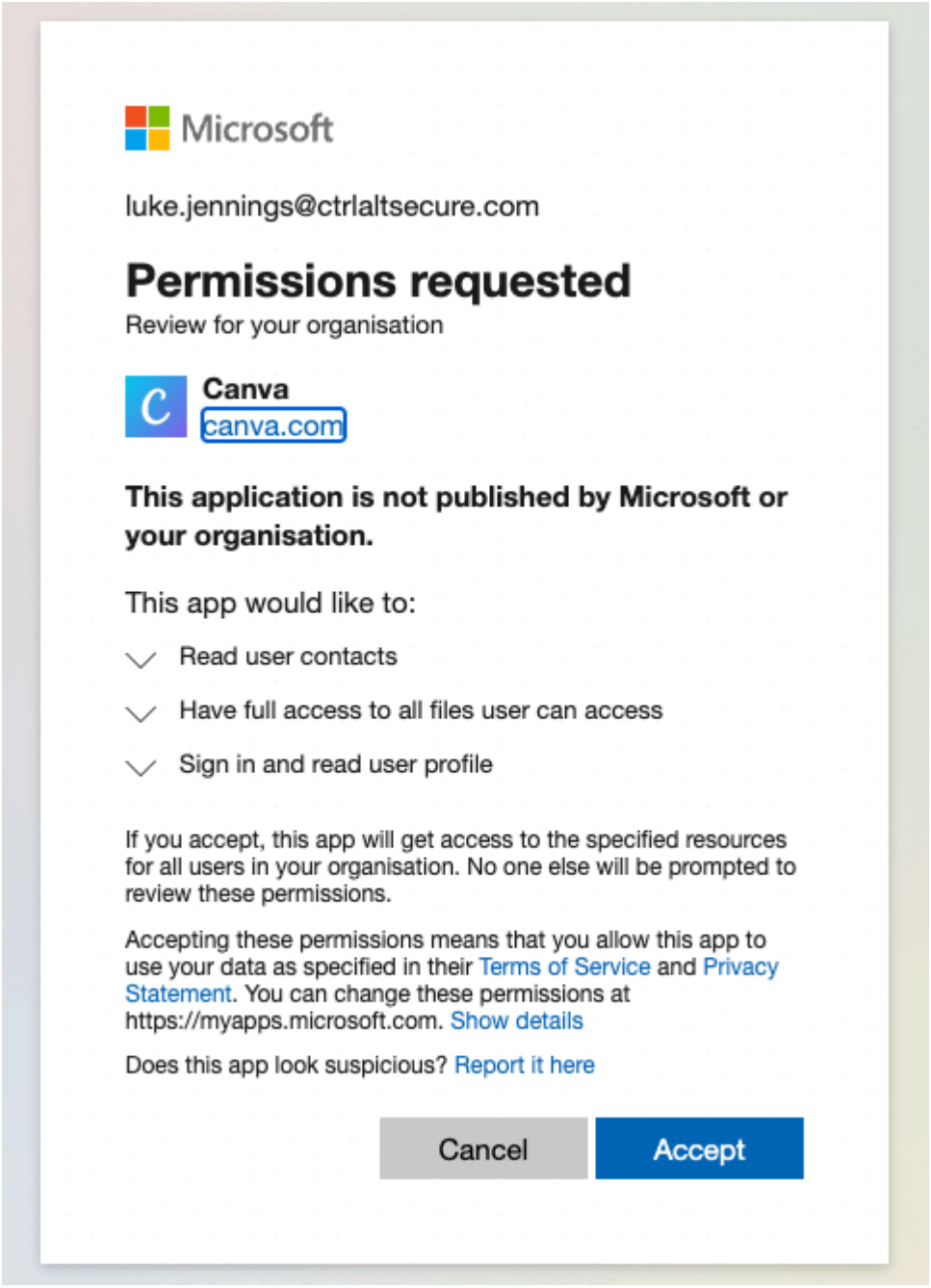
Attacks utilizing these types of techniques are becoming increasingly common and Microsoft even wrote about some real-world attacks they uncovered recently that involved the use of malicious OAuth apps.

OAuth attack #2: SaaS platform integration

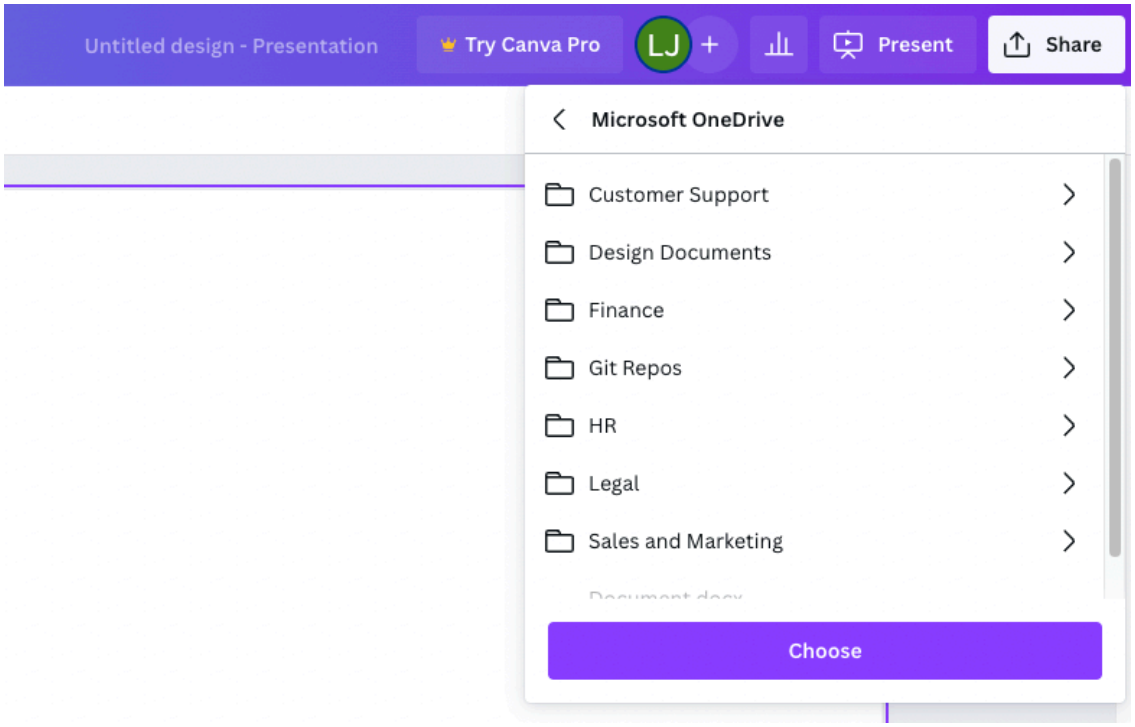
A similar approach to using a custom OAuth app is to make use of legitimate SaaS services that allow an attacker to make sensitive integrations as a more hide-in-plain-sight approach. For example, let's take the popular SaaS platform Canva, a graphic design tool that is used to create social media graphics, presentations, posters, documents and other visual content, as an example. Canva, like many SaaS platforms, allows you to make integrations with document stores like OneDrive and Google Drive in order to easily import and export files between Canva and them. If an attacker is interested primarily in maintaining access to a user's files, then they could make an integration with a platform like Canva and then use that to maintain access.

While this doesn't provide any raw capabilities beyond a custom OAuth app, an attacker may be more likely to go undetected in this scenario. Discovering an integration with a completely unknown, unverified OAuth app that hasn't been seen in use elsewhere in the organization, or anywhere at all, is suspicious. Finding an integration with a major SaaS platform, particularly if it is one in use by other users in the organization, is much less suspicious. Additionally, many of them will have verified ticks having been through Microsoft's or Google's own verification processes. The only downside for an attacker is having to find SaaS platforms that request the correct permissions and provide the functionality that the attacker is looking for, whereas a custom OAuth app could be used to request any permissions and code could be written to use those permissions however an attacker would like.

If a custom OAuth app is the equivalent of a custom implant on an endpoint, then using a legitimate SaaS platform integration is the equivalent of a more living-off-the-land approach, such as using TeamViewer, RDP or Powershell, etc.



Abusing a legitimate app to create custom permissions for data access



Using Canva to gain access to OneDrive

OAuth attack #3: Legitimate desktop/mobile app impersonation

Ok, we promise this is the last OAuth variation example - but it's another interesting way to abuse OAuth connections! Previously, we spoke of either connecting a custom OAuth app or using an OAuth integration via a legitimate SaaS platform. A custom OAuth app has the most flexibility for an attacker, but looks far more suspicious if discovered, whereas a legitimate SaaS platform looks much more….well, legitimate!

What if you could have both of those advantages in one? Well, that can be achieved, too! The reason SaaS platforms don't have the same flexibility is because they keep their client IDs and secrets for their apps so the attacker can only use the OAuth app indirectly via the features provided by the SaaS platform. However, some OAuth connections are made using desktop or mobile apps that obviously can't keep their OAuth app secrets secret from a user. While it is generally not possible for an attacker to make use of these in a consent phishing attack, due to not controlling the reply URLs, they can be used in a pure persistence scenario with an already compromised account.

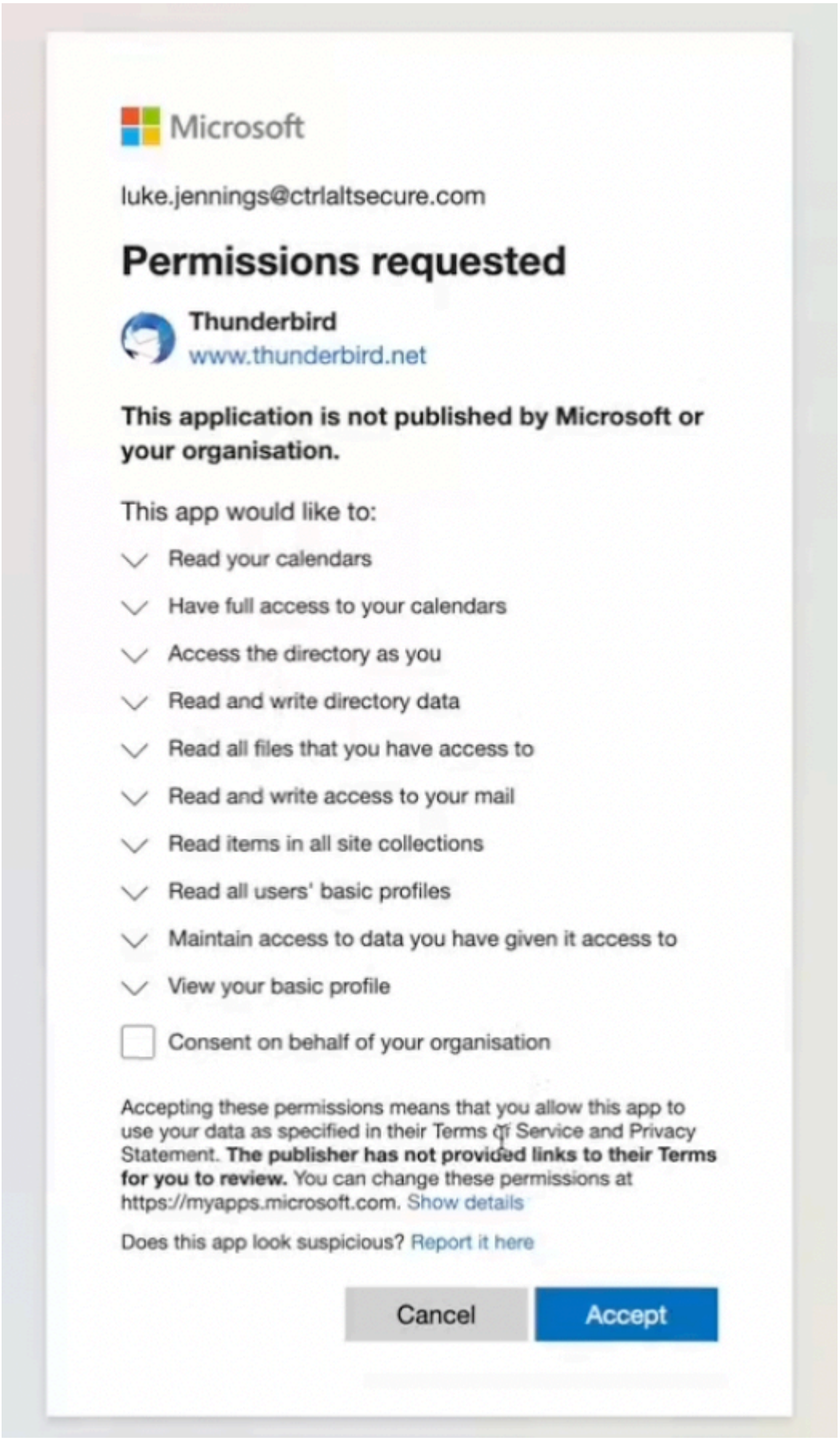
Let's take Mozilla Thunderbird, a cross-platform email client, as an example. The client IDs and secrets for different OAuth apps are actually stored in the source code in this case:

```
/**
 * Map of issuers to clientId, clientSecret, authorizationEndpoint, tokenEndpoint.
 * Issuer is a unique string for the organization that a Thunderbird account
 * was registered at.
 *
 * For the moment these details are hard-coded, since dynamic client
 * registration is not yet supported. Don't copy these values for your
 * own application - register one for yourself! This code (and possibly even the
 * registration itself) will disappear when this is switched to dynamic
 * client registration.
 */
var kIssuers = new Map([
  [
    "accounts.google.com",
    {
      clientId:
        "406964657835-aq8lmia8j95dhl1a2bvharmfk3t1hgqj.apps.googleusercontent.com",
      clientSecret: "kSmqreRr0qwBWJgbf5Y-PjSU",
      authorizationEndpoint: "https://accounts.google.com/o/oauth2/auth",
      tokenEndpoint: "https://www.googleapis.com/oauth2/v3/token",
    },
  ],
```

Thunderbird stores client IDs and secrets for different OAuth apps in the source code

As an attacker, this gives us multiple advantages.

- **App Impersonation** - These are client IDs that will be seen in use legitimately by other users and we can impersonate them. In Thunderbird's case, the Microsoft app isn't actually a verified app but the Google one shows as verified. Whatever the case, it looks much less suspicious than a completely unknown app with no known business use case.
- **Flexible Use** - We have access to the client IDs and secrets, so we can do whatever we want with the OAuth integration, writing custom code to query APIs as we please. We are not limited to the functionality provided by Thunderbird itself.
- **Arbitrary Permission Granting** - We aren't actually limited to just the permissions that Thunderbird would normally request (e.g. email/calendar). Since we're in control of the OAuth secrets, we can just request whatever scopes we want. For example, shown below is us using the Microsoft Thunderbird OAuth secrets to request permissions that also include access to all files, Sharepoint, AD access, etc.

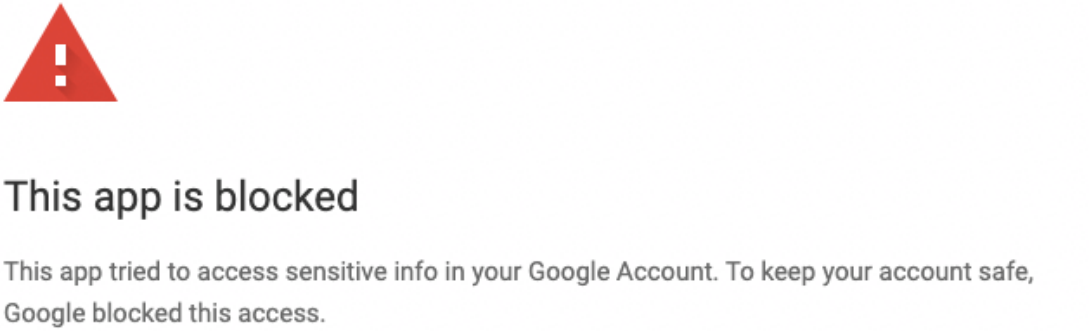


Abusing Thunderbird for arbitrary permission granting

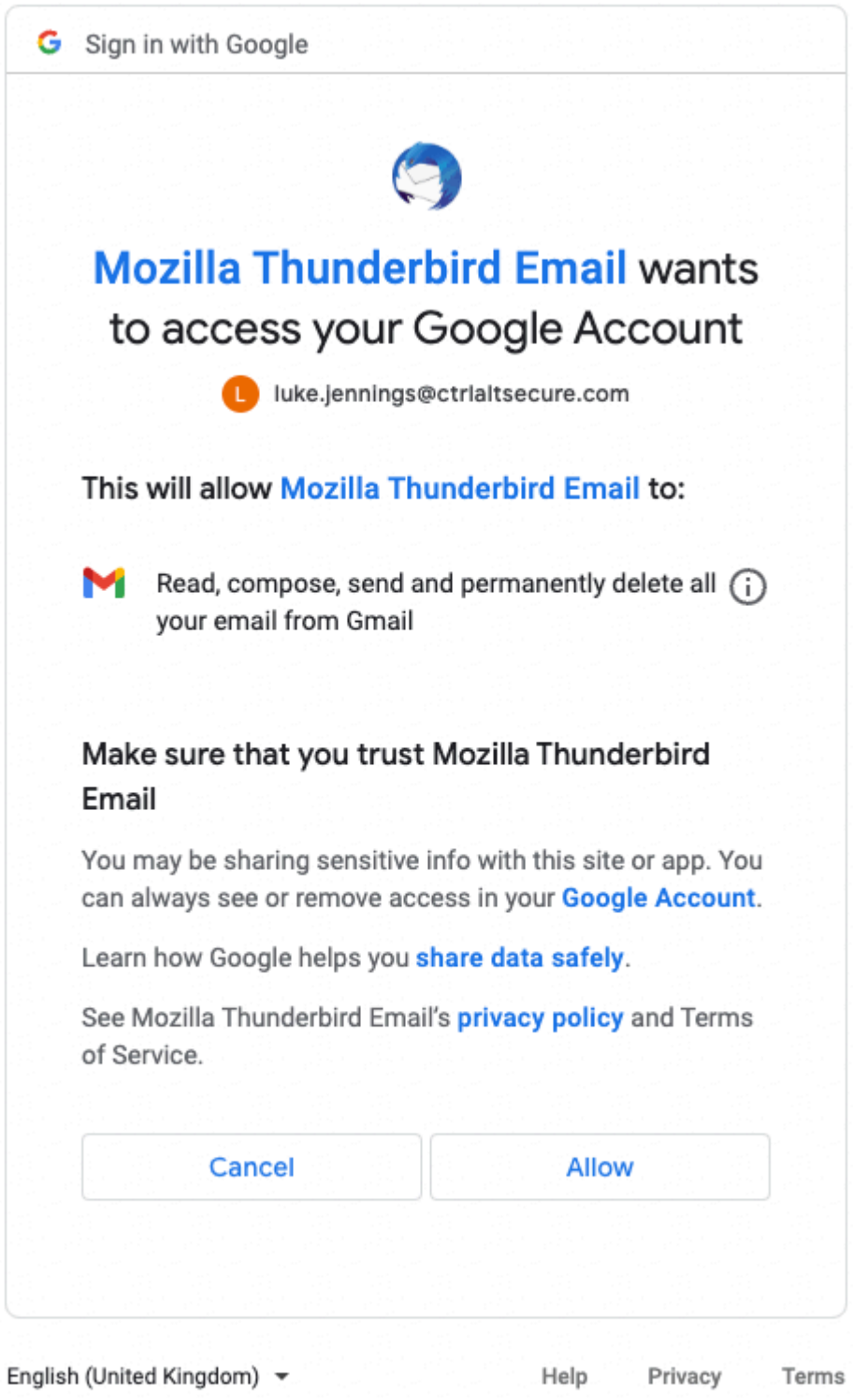
```
127.0.0.1 - - [21/Nov/2022 12:25:23] "GET /graphcall HTTP/1.1" 200 -
[*] Using token for 08162f7c-0fd2-4200-a84a-f25a4db0b584 (Thunderbird)
[*] Scopes available - Calendars.Read Calendars.ReadWrite Directory.AccessAsUser.All Directory.ReadWrite.All Files.Read.All
IMAP.AccessAsUser.All Mail.ReadWrite openid POP.AccessAsUser.All profile Sites.Read.All SMTP.Send User.ReadBasic.All email
[*] Enumerating OneDrive root folder
[*] Calling https://graph.microsoft.com/v1.0/me/drive/root/children
(Folder)        Customer Support
(Folder)        Design Documents
(Folder)        Finance
(Folder)        Git Repos
(Folder)        HR
(Folder)        Legal
(Folder)        Sales and Marketing
(File)          Document.docx
127.0.0.1 - - [21/Nov/2022 12:25:23] "GET /graphcall HTTP/1.1" 200 -
```

We're able to grant access to whatever we want, not just email

- **(Semi-)Bypass Google Restricted Scopes** - When it comes to arbitrary permission granting, there is a caveat with Google in that some of the more sensitive scopes Google offer are only available to selected approved and verified apps. Therefore, we can't necessarily just request access to any permission with Google. For example, if we modify Thunderbird to request access to Google Drive (a restricted scope) then we get the following:

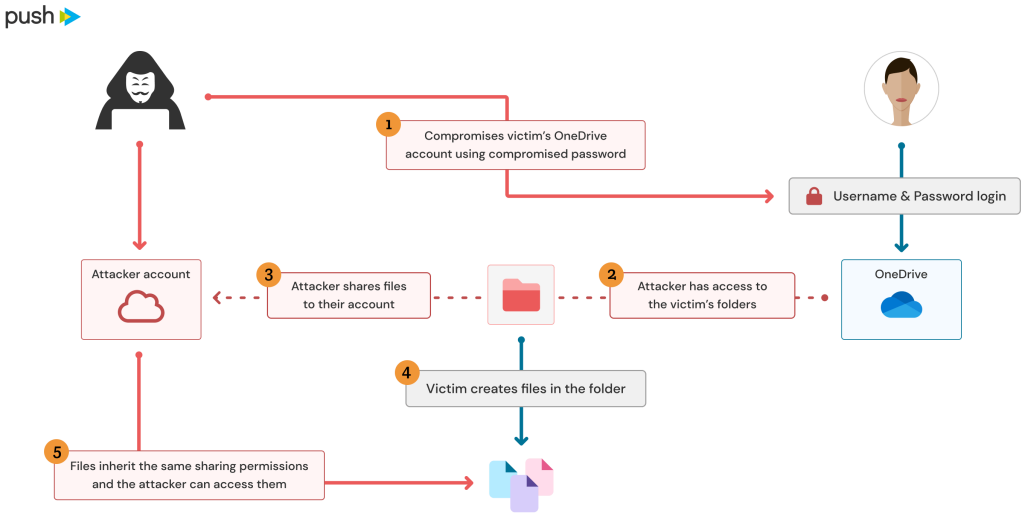


Access to Gmail is also considered a restricted scope. However, obviously Thunderbird is an email client, so if it uses OAuth it's going to want access to Gmail, right? Well, yes, the Thunderbird app ID is permitted access to Gmail data, so we can use it to gain that access and appear as a legitimate verified app, in addition to requesting any other non-restricted permissions we're interested in:



Using the Thunderbird app to gain access to Gmail

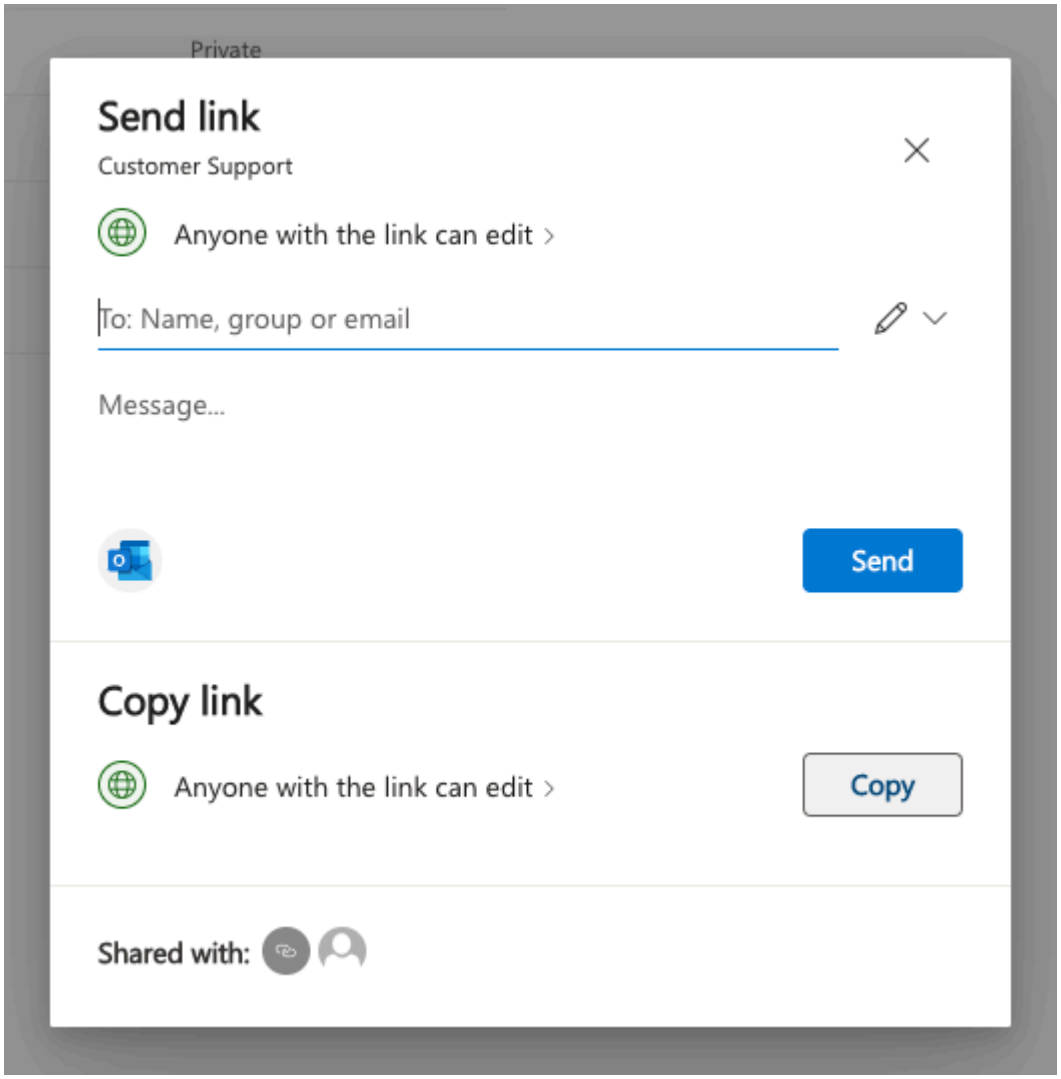
Document-sharing links



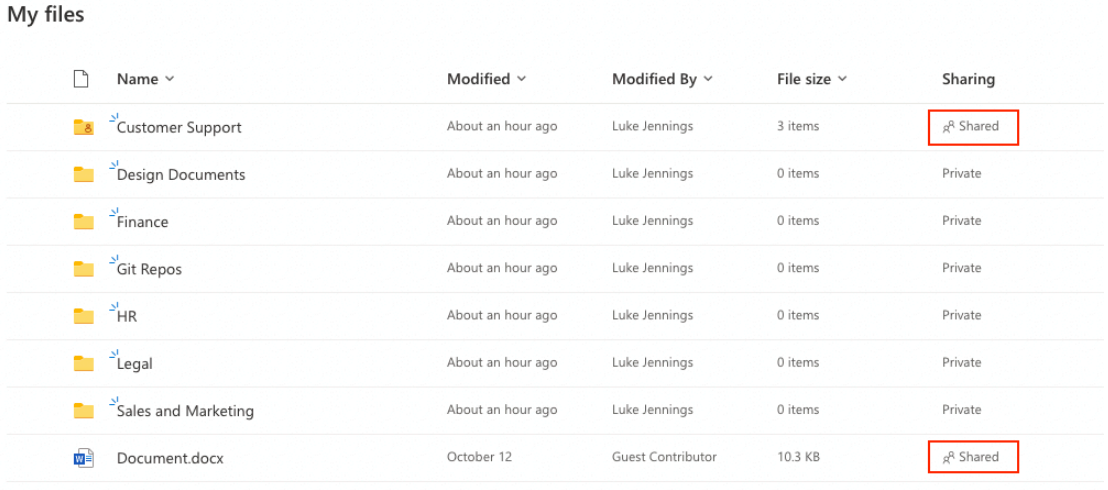
An attacker abusing document-sharing links to get persistent access

Ok, no more OAuth, we promise! The final option we want to highlight is the (ab-)use of document-sharing links. Many organizations make use of OneDrive, Sharepoint and Google Drive for document editing, sharing and collaboration. However, it's pretty common to want to share documents with people outside your organization sometimes too, right? That's where document-sharing links come in. You can create a document sharing link to share with specific individuals in other Google/Azure organizations or you can create anonymous links that anyone with knowledge of the (unguessable randomized) link can access.

Very similar functionality is present in both OneDrive and Google Drive, but this same legitimate functionality can also be abused by attackers to maintain backdoor access to either select files or entire root folders. Sharing a root folder will cause future files to inherit those sharing permissions. This is a modern repeat of the age-old problem of access control list (ACL) management on internal file servers, only now internet-based attackers can potentially abuse this without needing VPN or similar access.



Abusing OneDrive document-sharing functionality



OneDrive files now shows shared status

Conclusion

We've demonstrated a few new persistence options attackers are using against organizations as they move to the cloud. While some existing persistence and lateral movement options are no longer working in these environments, attackers have been able to quickly adapt to new conditions to get at their targets.

Some of these attacks have already been seen in the wild and others may already be happening under the radar. In any case, being aware of how attackers will try to compromise SaaS-first organizations helps you prepare to defend and respond to these attacks.

It’s extremely important for incident response teams to adapt to these changes, as a password reset and a device wipe is not sufficient to regain control of a user account, even when no lateral movement to internal systems has been performed.

New steps need to be added to IR playbooks in the event of user or device compromises to cover the revocation of OAuth permissions and refresh tokens, the auditing of mail rules and changes to document sharing configurations.
