Compromised Microsoft Key: More Impactful Than We Thought

By Shir Tamari

Published: 2023-07-21 · Archived: 2026-04-05 17:35:00 UTC

Microsoft and CISA recently disclosed a security incident impacting multiple customers of Exchange Online and Outlook.com. According to Microsoft, this incident stemmed from a threat actor attributed to China, Storm-0558, acquiring a private encryption key (MSA key) and using it to forge access tokens for Outlook Web Access (OWA) and Outlook.com. Additionally, the threat actor reportedly exploited two security issues in Microsoft's token verification process.

Microsoft have said that Outlook.com and Exchange Online were the only applications known to have been affected via the token forging technique, but Wiz Research has found that the compromised signing key was more powerful than it may have seemed, and was not limited to just those two services. Our researchers concluded that the compromised MSA key could have allowed the threat actor to forge access tokens for multiple types of Azure Active Directory applications, including every application that supports personal account authentication, such as SharePoint, Teams, OneDrive, customers' applications that support the "login with Microsoft" functionality, and multi-tenant applications in certain conditions.

In addition, while Microsoft mitigated this risk by revoking the impacted encryption key and publishing attacker IOCs, we discovered that it may be difficult for customers to detect the use of forged tokens against their applications due to lack of logs on crucial fields related to the token verification process.

Why is it so impactful? Identity provider's signing keys are probably the most powerful secrets in the modern world. For example, they are much more powerful than TLS keys. Even if an attacker got access to the google.com TLS key, they would still need to somehow impersonate a google.com server to gain significant impact. With identity provider keys, one can gain immediate single hop access to everything, any email box, file service or cloud account. This isn't a Microsoft specific issue, if a signing key for Google, Facebook, Okta or any other major identity provider leaks, the implications are hard to comprehend. Our industry – and especially cloud service providers – must commit to a greater level of security and transparency concerning how they protect critical keys such as this one, to prevent future incidents and limit their potential impact.

In this post, we will share how we were able to confirm which private key was acquired by the threat actor and how we determined its permissions. We will also unpack some of the technical aspects of this incident and help detect potential use of this compromised key within your environments.

Compromised consumer signing key – who are you?

On July 11th, 2023, Microsoft revealed that a malicious actor had obtained an MSA consumer signing key, allowing them to forge access tokens for Exchange Online and Outlook.com accounts.

Determined to learn more about the incident, we launched an investigation.

First, we checked which keys could sign OpenID tokens for Microsoft accounts and Azure Active Directory applications. We therefore examined Microsoft's official documentation for OpenID token verification. Interestingly, we discovered that all Azure personal account v2.0 applications depend on a list of 8 public keys, and all Azure multi-tenant v2.0 applications with Microsoft account enabled depend on a list of 7 public keys (at the time of writing).

Using the Internet Archive's Wayback Machine, we noticed that one of the listed public keys that had been present since at least 2016 was replaced sometime between June 27th and July 5th, 2023, matching the time frame in which Microsoft replaced the acquired key according to their blog post.

Metadata of the public key replaced between June 27th and July 5th

The old public key's certificate revealed it was issued on April 5th, 2016, and expired on April 4th, 2021, and its thumbprint matched the thumbprint of the key Microsoft listed in their latest blog post, named "Thumbprint of acquired signing key":

> The decoded certificate of the old key (1LTMzakihiRla_8z2BEJVXeWMqo). Obtained from the list intended June 27th, 2023 version of the certificate list for Azure common (mixed audience) applications.

This led us to believe that although the compromised key acquired by Storm-0558 was a private key designed for Microsoft's MSA tenant in Azure, it was also able to sign OpenID v2.0 tokens for multiple types of Azure Active Directory applications.

What is the significance of a compromised OpenID signing key?

The Azure identity platform publishes multiple lists of trusted keys scoped to different application types. These serve to validate the integrity of tokens which are issued by Azure Active Directory (AAD). During the authentication process for an AAD application, the application must confirm the token's authenticity by verifying its signature against the correct trusted public key list. This verification determines whether the token should be trusted.

Azure Active Directory multi-tenant applications:

Azure Active Directory public certificates' lists

If any of the keys from one of these lists are compromised, there is a significant risk for applications using that list for validation. Such a compromise could enable unauthorized parties to forge valid access tokens for consumption by any application that depends on the Azure identity platform under certain conditions (see below).

> The risks of compromised OpenID signing key

Based on what we can deduce from Microsoft's blog post, Storm-0558 seemingly managed to obtain access to one of several keys that were intended for signing and verifying AAD access tokens. The compromised key was trusted to sign any OpenID v2.0 access token for personal accounts and mixed-audience (multi-tenant or personal account) AAD applications.

The types of applications that could trust the key acquired by Storm-0558

In other words, Storm-0558 could have theoretically used the private key it acquired to forge tokens to authenticate as any user to any affected application that trusts Microsoft OpenID v2.0 mixed audience and personal-accounts certificates.

Which applications are affected?

Based on our analysis, only Azure Active Directory applications that work with Microsoft's OpenID v2.0 were affected. Version 1.0 applications were not using the compromised key for token validation and therefore were not affected.

Applications supporting Personal Microsoft accounts only

Any Azure Active Directory application that supports "Personal Microsoft accounts only" and works against Microsoft's v2.0 protocol was affected**. This includes** managed Microsoft applications, such as Outlook, SharePoint, OneDrive, and Teams, as well as customers' applications that support Microsoft Account authentication, including those who allow the "Login with Microsoft" functionality.

Applications supporting accounts in any organizational directory (Any Azure AD directory – Multi-tenant) and personal Microsoft accounts (e.g. Skype, Xbox)

Any Azure Active Directory application that supported "mixed audience" and works against Microsoft's v2.0 protocol was affected as well. The threat actor could forge valid access tokens and impersonate application users who signed in with their Personal Microsoft account.

To restrict the power of MSA keys in impersonating organizational accounts, Microsoft introduced an extension to the OpenID protocol. This extension advises developers to validate the issuer claim by comparing it with the issuer field in the list of the OpenID public keys. By doing this, it aims to prevent an MSA key from signing access tokens with an issuer different than the MSA tenant (9188040d-6c67-4c5b-b112-36a304b66dad). This extension is specific to Microsoft and the responsibility of its implementation rests with the application owner. Therefore, there is a concern that many applications lack this procedure and as a result, the threat actor could potentially impersonate organizational accounts as well (according to Microsoft's blogpost, OWA was affected by a similar issue).

To assist Azure developers with adopting this validation functionality, Microsoft added it to their official Azure SDK on July 12.

Applications supporting accounts in any organizational directory (Any Azure AD directory – Multi-tenant)

IIf the multi-tenant application is configured to rely on the "common" v2.0 keys endpoint (instead of "Organizations"), then it is affected but also should be considered misconfigured. The official Microsoft documentation is not clear on when the "common" endpoint should be used, and therefore, some multi-tenant applications could be affected as well.

Applications supporting accounts in this organizational directory only (Singletenant)

Single tenant applications were not affected.

> How different types of users may have been affected depending on the application type and whether it was properly validating access tokens

How does key forging work?

OpenID keys are fundamentally JWTs signed by an authorized private key. As part of the Azure Active Directory token validation procedure, the app developer must confirm that the key is indeed signed by the relevant authority for the intended scope, and that the token's `aud` field matches the targeted application's scope.

To confirm whether the token was truly signed by a trusted Azure authority, the application developer queries a metadata endpoint (named `jwks_uri`) to pull the permitted certificates for signature verification and verify the token against it.

To forge a valid access token, the threat actor could have crafted a JWT token, populated it with a victim's data (e.g. email address), and finally signed it with the trusted compromised key that is listed under the Azure Active Directory public certificates' endpoint. By submitting the signed token to a targeted application, the malicious actor could have then impersonated the victim.

Here is a fictitious example of such a forged OpenID token signed by the compromised encryption key, `1LTMzakihiRla_8z2BEJVXeWMqo`:

According to Microsoft's guidelines, in order for the token to be considered valid, the issuer claim (`iss`) must be set to https: //sts.windows.net/9188040d-6c67-4c5b-b112-36a304b66dad/v2.0 since it was specified in the issuer field within the `jwks_uri` endpoint. As for the tenant ID claim (`tid`), it must accordingly be set to `9188040d-6c67-4c5b-b112-36a304b66dad`, the MSA tenant's ID.

For AAD mixed-audience applications (multi-tenant and personal-account), any token signed by the MSA tenant for an Azure AD account could be deemed valid, as long as it impersonates a personal account.

For additional details, check out Microsoft's official guidelines on how to verify ID Tokens.

Are Azure customers still at risk?

Due to Microsoft's revocation of the compromised key, Azure Active Directory applications will no longer accept forged tokens as valid tokens. Tokens with extended expiration dates will also be rejected by these applications.

However, during previously established sessions with customer applications prior to the revocation, the malicious actor could have leveraged its access to establish persistence. This could have occurred by leveraging the obtained application permissions to issue application-specific access keys or setting up application-specific backdoors. A notable example of this is how, prior to Microsoft's mitigation, Storm-0558 issued valid Exchange Online access tokens by forging access tokens for Outlook Web Access (OWA).

There is another potential risk to applications that retained copies of the AAD public keys prior to Microsoft's certificate revocation. Applications that rely on local certificate stores or cached keys and still trust the compromised key remain susceptible to token forgery. It is imperative for these applications to immediately refresh the list of trusted certificates. Microsoft advises refreshing the cache of local stores and certificates at least once a day.

Recommendations for Azure users

To identify whether a compromised key was used in your environment, identify all potentially affected applications in your environment, search for forged tokens usage (as explained in the next section) and leverage the Indicators of Compromise (IoCs) published by Microsoft on their blog to look for any activity that originates from the IP addresses provided by Microsoft.

In addition, make sure that none of the applications use a cached version of the Microsoft OpenID public certificates, and if so, refresh the cache.

Microsoft has added additional verifications to the official Azure SDK, which are designed to prevent the use of MSA keys to authenticate to organization accounts. Users of the package are advised to update it to the latest version.

How to detect the compromised key in your environment

Since the threat actor can forge access tokens offline, there is no trail in the Azure portal for token issuance. The only way for cloud customers to identify whether the key was used to target their apps or users is by reviewing application-specific logs for potentially affected AAD apps. Therefore, application owners who want to protect their systems will have to check whether a forged token has been used against their applications.

To the best of our knowledge, the only affected applications were those that utilized Microsoft v2.0 access token verification using the endpoints ”https://login.microsoftonline.com/common/discovery/v2.0/keyscommon“ and “https://login.microsoftonline.com/consumers/discovery/v2.0/keys“. These parameters make it feasible to filter out applications that were not exposed to this issue.

First, to identify which AAD applications in your environment might be affected, you can run the following Azure CLI command:

Additionally, your AAD applications might also be associated with Azure WebApps. To identify which AAD apps are redirecting to any of your WebApps, you can run the following CLI command:

Next, to identify potentially malicious activities in applications, it is necessary to examine suspicious authentication attempts via OpenID tokens signed by the compromised key. This can be done by unpacking the access tokens used against the application and searching for the string `1LTMzakihiRla_8z2BEJVXeWMqo` within the `kid` field of the JOSE Header.

According to Microsoft, the compromised key was inactive and therefore any access token signed by this key must be considered suspicious.

Unfortunately, there is a lack of standardized practices when it comes to application-specific logging. Therefore, in most cases, application owners do not have detailed logs containing the raw access token or its signing key. As a result, identifying and investigating such events can prove exceedingly challenging for app owners.

When examining an AAD application configured solely for multi-tenant authentication (without support for Microsoft personal accounts), it is possible to detect forged tokens by filtering for `iss` and `tid` claims within the access token. Applications commonly use these fields and they are more likely to be present in application logs. Moreover, any attempt to connect with an access token signed by the MSA tenant ID `9188040d-6c67-4c5b-b112-36a304b66dad` may indicate the use of a compromised key.

Finally, if you've enabled HTTP Logs in your WebApp, you might be able to see which IP addresses have accessed your application. Based on Microsoft's blogpost, the following IP addresses are associated with the threat actor, so you should validate if your WebApp might have been impacted by running the following query in Log Analytics for each of your potentially affected Web Apps:

For additional guidance on searching for signs of persistence in your environment, see our "CircleCI Incident Sign of Persistence" blog.

Key Takeaways

The full impact of this incident is much larger than we Initially understood it to be. We believe this event will have long lasting implications on our trust of the cloud and the core components that support it, above all, the identity layer which is the basic fabric of everything we do in cloud. We must learn from it and improve.

At this stage, it is hard to determine the full extent of the incident as there were millions of applications that were potentially vulnerable, both Microsoft apps and customer apps, and the majority of them lack the sufficient logs to determine if they were compromised or not. However there are some critical actions items that application owners should perform. The first and foremost is to update their Azure SDK to the latest version and ensure their application cache is updated, otherwise their apps may still be vulnerable to a threat actor using the compromised key.

We will continue to closely monitor this incident and provide updates; this is still an ongoing investigation and there are many unanswered questions (how did the threat actor acquire the key? When exactly did it happen? Were other keys compromised as well?). Finally, we want to thank the Microsoft team for working closely with us on this blog and helping us ensure it is technically accurate.

References

- https://msrc.microsoft.com/blog/2023/07/microsoft-mitigates-china-based-threat-actor-storm-0558-targeting-of-customer-email/
- https://www.cisa.gov/news-events/cybersecurity-advisories/aa23-193a
- https://blogs.microsoft.com/on-the-issues/2023/07/11/mitigation-china-based-threat-actor/
- https://www.microsoft.com/en-us/security/blog/2023/07/14/analysis-of-storm-0558-techniques-for-unauthorized-email-access/
- https://github.com/AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet/pull/2136/files
- https://github.com/MicrosoftDocs/azure-docs/commit/f17445bb9202a89964ea7311c4374806adfcb28c

---