OASIS

AuthQuake

A practical, low profile, MFA bypass

Tal Hason
Research Engineer

Published on
December 11, 2024

Summary

Oasis Security's research team uncovered a critical vulnerability in Microsoft's Multi-Factor Authentication (MFA) implementation, allowing attackers to bypass it and gain unauthorized access to the user's account, including Outlook emails, OneDrive files, Teams chats, Azure Cloud, and more. Microsoft has more than 400 million paid Office 365 seats, making the consequences of this vulnerability far-reaching.

The bypass was simple: it took around an hour to execute, required no user interaction and did not generate any notification or provide the account holder with any indication of trouble.

Upon discovery, Oasis reported the flaw to Microsoft and collaborated with them to resolve it. Below are details of the vulnerability, its resolution, and lessons learned.

Multi-factor authentication (MFA) - numbers and trends

Authentication relying only on “something you know” - like a password or a PIN - is vulnerable as passwords can be guessed, stolen, or exposed through phishing attacks.
By requiring more authentication factors, MFA significantly improves your authentication security. Additional factors can be “something you have” (e.g a mobile device), “something you are” (e.g biometrics) or “something you do” (e.g a gesture or pattern).

MFA is a recommended practice endorsed by both identity providers and security experts - according to Microsoft (1, 2, 3, 4), an account requiring MFA “is more than 99.9% less likely to be compromised”. OWASP’s MFA Cheat Sheet advocates MFA as well - “MFA is by far the best defense against the majority of password-related attacks, including brute-force, credential stuffing and password spraying”.

We often hear that the lack of MFA makes it an attractive target for attackers. For example, this June Snowflake customers were targeted, a threat actor exploiting leaked credentials from Snowflake accounts lacking MFA used them to exfiltrate private records of over 500 million individuals. SnowFlake has defaulted to MFA since.

Recognizing the need to reduce security risks and following expert recommendations, organizations have increasingly implemented MFA as a critical security measure to protect against unauthorized access. A report by Okta highlights the widespread adoption of MFA - “64% among Okta workforce users, while at least 90% of administrators use MFA.”

Now add that, beyond being a recommendation, MFA may also be a requirement for organizational compliance. For example PCI DSS requires 2FA for remote access (section 8.3), ISO 27001 mandates to adopt methods that protect against brute-force attempts (section A.9.4.2) and in SOC2, MFA is a commonly recommended control within the "Privacy" Trust Service Criteria.

Lastly - there are various methods utilizing MFA. According to a JumpCloud 2024 survey “95% of employees using MFA do so via a software program, such as a mobile app” which we can assume refers to the mobile authenticators (aka TOTP Authenticator apps, MFA apps, 2FA apps). Mobile authenticators are usually implemented according to RFC-6238 and generate 6-digit time-limited passwords for each configured service, which users submit as a second factor during authentication.

To sum it all up - **MFA is widely adopted, particularly through mobile authenticators, to the extent that its reliability is often taken for granted. But what happens when that assumption is compromised?**

MFA on Microsoft

Microsoft supports a variety of MFA methods - one of which is an MFA app, which is the one in discussion here.

When signing-in to Microsoft, a user supplies their email and password, selects one of the pre-configured MFA methods, then proceeds to supply the code to complete the authentication.

To understand the flaw, we must first discuss the HTTPS interaction going on between the user and the validator when authenticating-

1. User browses to https://login.microsoftonline.com or any resource or portal requiring authentication (e.g https://portal.azure.com)
2. The page sends A GET request to https://login.microsoftonline.com/organizations/oauth2/v2.0/authorize together with a few parameters describing the client software, and returns a list of values. Four of the returned values are kept and later used to identify this session - **canary, flowToken, sessionId** (128-bit GUID) and **ctx** (user app state and context).
3. A login page is rendered on the browser where users supply their login information. This, along with the previously specified session parameters, is being sent with a POST request to https://login.microsoftonline.com/common/login
4. If the credentials are valid, the user will be able to select the “Use a verification code” method which sends a POST request to https://login.microsoftonline.com/common/SAS/BeginAuth together with the session parameters previously specified. A new session ID is then generated and returned for use with the MFA attempts. One could create multiple such requests based on the same session parameters obtained from step 2.

Using the authenticator app, the user types in the 6-digit code and the code is sent via a POST to https://login.microsoftonline.com/common/SAS/EndAuth, which returns a success or fail response. Up to 10 consequent fail retries are supported for a single session.

The vulnerability

The limit of 10 consequent fails was only applied to the temporary session object, which can be regenerated by repeating the described process, with not enough of a rate limit.
Simply put - one could execute a lot of attempts simultaneously.

By rapidly creating new sessions and enumerating codes, the Oasis research team demonstrated a very high rate of attempts that would quickly exhaust the total number of options for a 6-digit code (1M).

Moreover, during this period, account owners do not receive any email or alert about the massive number of consequent failed attempts, which makes this attack very low profile.

As mentioned, authenticator app codes are time-limited, and so the next question was around the available time frame attackers had for guessing a single code.

RFC-6238 is the TOTP guideline, suggesting a different code be generated for each time-frame of 30 seconds-

“We RECOMMEND a default time-step size of 30 seconds...as a balance between security and usability.” and indeed most validators use this setting.

However, due to potential time differences and delays between the validator and the user, the validator is encouraged to accept a larger time window for the code, again from the RFC-6238 guidelines-

“The validation system should compare OTPs not only with the receiving timestamp but also the past timestamps that are within the transmission delay. A larger acceptable delay window would expose a larger window for attacks. We RECOMMEND that at most one time step is allowed as the network delay.”

This means that a single TOTP code may be valid for more than 30 seconds.

Before testing the length of this grace period on Microsoft, a quick search led us to this post from a Microsoft employee suggesting the period varies and can last up to 5 minutes.

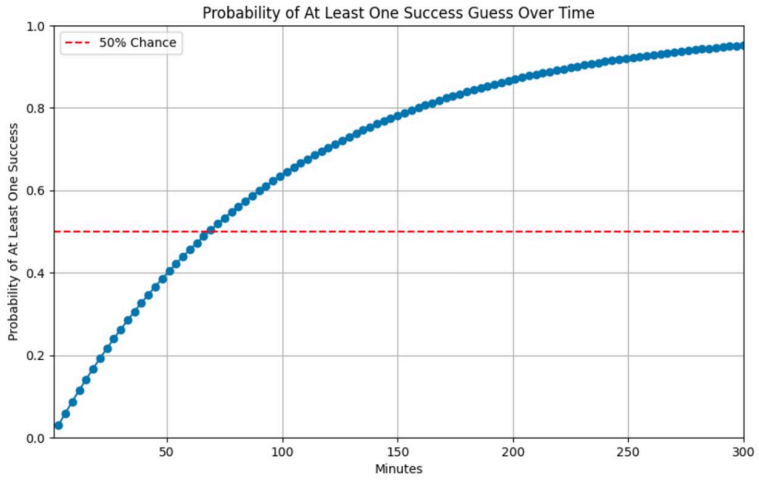
Our testing with Microsoft sign-in has shown a tolerance of around 3 entire minutes for a single code, extending 2.5 min past its expiry, allowing 6x more attempts to be sent.

Given the allowed rate we had a 3% chance of correctly guessing the code within the extended timeframe.

A malicious actor is likely to proceed and run further sessions until they hit a valid guess; the Oasis Research team didn't run into any issues or limitations doing that.

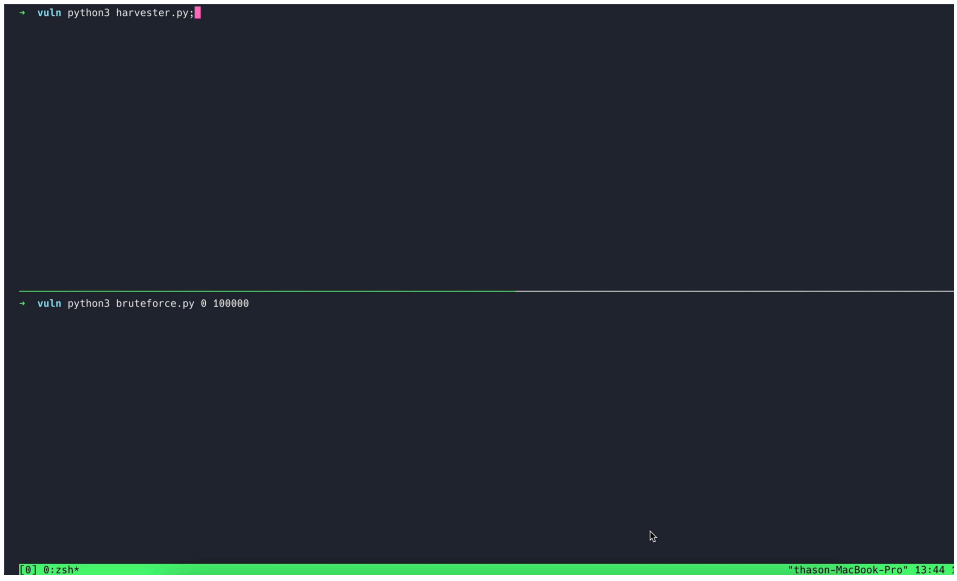
After 24 such sessions (~70 minutes) a malicious actor would already pass the 50% chance of hitting a valid code.

What's more, this is before considering the additional codes generated within the timeframe that would make a few more guessed codes valid.



After 70 minutes we are already past 50% chance of hitting a valid code

We have successfully attempted this method multiple times, below is a screen recording (click on the image to view the full clip) of one such successful attempt, where the code was guessed early on -



The fix

While specific details of the changes are confidential, we can confirm that Microsoft introduced a much stricter rate limit that kicks-in after a number of failed attempts, the strict limit lasts around half a day.

Vulnerability timeline and Microsoft response

- 24/06/2024 - Microsoft Acknowledgment of the issue
- 04/07/2024 - Microsoft Deployed a temporary fix
- 09/10/2024 - Microsoft Deployed Permanent Fix

Guidelines for organizations using MFA

We encourage you to use MFA, either Authenticator apps or stronger passwordless methods; the flaw discussed in this article belongs to a specific implementation that has been fixed prior to releasing this text.

Even with MFA enabled, it's crucial to stay vigilant by monitoring for leaked credentials and changing your password regularly. While MFA can protect you if your credentials are compromised, as shown here, an implementation flaw in the validation process can quickly render it ineffective.

Add a mail alert for failed MFA attempts.
In general, monitoring for wrong password sign-ins is likely to create a lot of noise, especially on targeted accounts. However, filtering this specifically to failed second factor codes reduces the noise to only those cases where the actor holds a correct password, which are much more interesting. In addition, the alerts should be sent to the account owner, who can verify whether the login attempts were made by them or not, allowing them to take immediate action if necessary.

Guidelines for MFA implementations

Introducing rate-limits and making sure they are properly implemented is crucial.
Rate limits might not be enough, in addition - **consequent failed attempts should trigger an account lock.**
NIST Special Publication 800-63 guides for such implementation on section 4.3.3 (Authentication Process)-
“Additionally, mechanisms located at the verifier can mitigate online guessing attacks against lower entropy secrets — like passwords and PINs — by limiting the rate at which an attacker can make authentication attempts, or otherwise delaying incorrect attempts. Generally, this is done by keeping track of and limiting the number of unsuccessful attempts, since the premise of an online guessing attack is that most attempts will fail.”

Notifying users on failed authentications is also suggested, consider this guideline from OWASP’s MFA Cheat Sheet-

“When a user enters their password, but fails to authenticate using a second factor...steps that should be taken when this occurs:

- Notify the user of the failed login attempt, and encourage them to change their password if they don't recognize it.
- The notification should include the time, browser and geographic location of the login attempt.
- This should be displayed next time they login, and optionally emailed to them as well.”

When measuring the effectiveness of the rate-limit **one must factor the entire time-frame for a valid code**, and perhaps consider further limiting that time-frame. Quoting once again from RFC-6238-
“We RECOMMEND that at most one time step is allowed as the network delay.”

The Snowflake attack mentioned earlier in the post, highlights that in addition to implementing MFA, vendors should actively help its adoption, for example - **allow admins to enforce MFA for all users.**

About Oasis Security

Oasis Security is the management and security solution for non-human identities. Purpose-built to address the unique challenges of visibility, security, & governance of NHIs across hybrid cloud environments.

With advanced AI-driven analytics, Oasis automatically discovers NHIs, assesses their risks, and identifies ownership across the entire environment. Its integrated, policy-based governance capabilities ensure seamless orchestration of the NHI lifecycle, including remediation and compliance management—all within a single, unified solution.

Leading organizations across a wide range of industries use Oasis to foster innovation and collaboration among security, identity, and engineering teams, enabling secure digital transformation and cloud adoption.

The Oasis Research team

Our dedicated research team is committed to enhancing security in the field of identity. We take pride in our responsible and professional collaboration with vendors to address vulnerabilities and strengthen overall security.

The authors

Tal Hason, Research Engineer at Oasis Security

Tal specializes in advanced security research, creating solutions to secure hybrid cloud environments from vulnerabilities in unmanaged non-human identities. With over six years of experience in cybersecurity, he has led teams to address complex challenges, leveraging his expertise in vulnerability research and threat intelligence to drive innovation and deliver robust security solutions. Tal is dedicated to advancing the field through meticulous analysis and innovative problem-solving.

Elad Luz, Research Lead at Oasis Security

Elad Luz has over 20 years of research experience in fields such as software vulnerabilities, reverse engineering, network protocol analysis, threat detection, and ML. Prior to Oasis, he served as a CDR Research Lead at Wiz and as the Head of Research at CyberMDX. He has publicly disclosed over 20 vulnerabilities, demonstrating a strong commitment to enhancing security across multiple platforms.