VECTRA

Common Vulnerabilities and Exposures

Undermining Microsoft Teams Security by Mining Tokens

September 13, 2022

Connor Peoples
SSPM Architect


ERMINING MICROSOI
IS SECURITY BY
NG TOKENS

Overview

In August 2022, the Vectra team identified a post-exploitation opportunity allowing malicious actors with sufficient local or remote file system access to steal valid user credentials from Microsoft Teams due to their plaintext storage on disk. This plaintext credential management was determined to impact all commercial and GCC Desktop Teams clients for Windows, Mac, and Linux.

While credential harvesting from memory is a common post-exploitation step, we believe that lowering the bar necessary to harvest creds down to just simple read access to the file system expands opportunities for an adversary, simplifies their task, and is particularly interesting when stolen credentials offer an opportunity to retain user access unencumbered by otherwise pesky Multi-Factor Authentication (MFA) speedbumps.

With these tokens, attackers can assume the token holder's identity for any actions possible through the Microsoft Teams client, including using that token for accessing Microsoft Graph API functions from an attacker's system. Additionally, these tokens are equally valid with MFA-enabled accounts, creating an allowance to bypass MFA checks during ongoing use.

Microsoft is aware of this issue but indicated it did not meet their bar for immediate servicing. Until Microsoft moves to update the Teams Desktop Application, we believe customers should be aware of the risks posed by plaintext token storage and mitigate that risk by instrumenting monitoring for unusual file access or modification to file system ACLs.

The Genesis of the Hunt

The investigation kicked off when a Vectra customer complained about how Microsoft Teams manages disabled identities. End users cannot remove deactivated accounts through the UI because the Teams application requires the account to be signed in to remove it from the client. Of course, users cannot do this when their user account is disabled. To help resolve this issue, we began to look at the local configuration data inside the Teams client and unravel how it works.

Electron – a Security Negative

Microsoft Teams is an Electron-based app. Electron works by creating a web application that runs through a customized browser. This is very convenient and makes development quick and easy.

However, running a web browser within the context of an application requires traditional browser data like cookies, session strings, and logs.

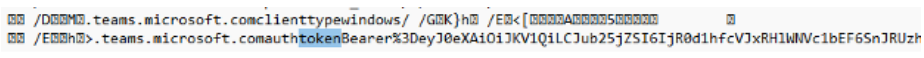
This is where the root of this issue lies as Electron does not support standard browser controls like encryption, and system-protected file locations are not supported by Electron out of the box but must be managed effectively to remain secure. Therefore, by default the way Electron works incentivizes creating overly transparent applications. Since Electron obfuscates the complexities of creating the application, it is safe to assume that some developers may be unaware of the ramifications of their design decisions and it is common to hear application security researchers bemoan the use of this framework due to critical security oversights.

Diving into the Structure

We first began to explore methods to expunge any references to the logged-in account(s). Our goal was to remove the old accounts and force Teams to operate as if they were gone. Multiple attempts to modify the configuration file and first-time run files amounted to nothing. As a stab in the dark, we searched for the known user principal name, and two critical files returned.

The first important file was an ldb file with access tokens in clear text. Upon review, it was determined that these access tokens were active and not an accidental dump of a previous error. These access tokens gave us access to the Outlook and Skype APIs. It is important to know that the Microsoft Teams architecture is a conglomeration of a wide variety of M365 services that relies on Skype, SharePoint, and Outlook to function—this explains the presence of these tokens.

The next file is a browser Cookies database like the "Cookies" we agree to on every website (thanks, GDPR). Cookies store data like session information, marketing tags, account information, and in some instances, access tokens. (un)Fortunately, the Teams Desktop application is storing the tokens here as well.



The best way to read the Cookie DB is to use a sqlite3 database client. With this client, we can extract just the values we need. The following query returns the token's name and the token value.

```
sqlite> SELECT name, value FROM Cookies WHERE name LIKE "%token%";
```

We evaluated each token against the Microsoft jwt validation service, https://jwt.ms. Each token we found was active and worked without requiring additional authentication. The realization began to dawn that the initial problem of having to re-install teams was a much smaller issue than the identity abuse opportunity potentially looming in the Microsoft Teams client.

Let's Do Something

The team set to work with this knowledge and began to craft tooling that took advantage of these unprotected credentials. After considering multiple options, it was determined that sending a message to the account of the credential holder through Teams with an access token would be appropriate. With this goal in mind, we launched the Teams client in the browser to track API calls when sending messages and found this gem:

https://amer.ng.msg.teams.microsoft.com/v1/users/ME/conversations/48:notes/messages

This API endpoint lets us send messages to ourselves, and we do not have to futz around with account enumeration. Next, we needed the access token. We used the SQLite engine. SQLite does not require installation, so the tooling downloads SQLite to a local folder and executes it to read the Cookies DB, where we extract the Skype Access token required for sending messages.

With the token in hand and our destination in mind, the last piece was to string together a message. The request body took some time to get working, but we eventually succeeded. We set the message to send with the high importance flag set, and the subject of "You've Been PWND." The message itself is the Skype Access Token.

The tool sends the message at this point, and we can validate that the access token is in our personal chat.

Not too shabby for a morning's work.

The Implications of Unsecured Credentials

Microsoft stores these credentials to create a seamless single sign-on experience within the desktop application. However, the implementation of these security choices lowers the bar.

Anyone who installs and uses the Microsoft Teams client in this state is storing the credentials needed to perform any action possible through the Teams UI, even when Teams is shut down. When these tokens are stolen, it enables attackers to modify SharePoint files, Outlook mail and calendars, and Teams chat files. Attackers can tamper with legitimate communications within an organization by selectively destroying, exfiltrating, or engaging in targeted phishing attacks.

The Big Scary –The Ultimate Phish

Here the thing that truly frightens us is the proliferation of post-MFA user tokens across an environment – it enables subsequent attacks that do not require additional special permissions or advanced malware to get away with major internal damage. With enough compromised machines, attackers can orchestrate communications within an organization. Assuming full control of critical seats–like a company's Head of Engineering, CEO, or CFO — attackers can convince users to perform tasks damaging to the organization. How do you practice phish testing for this?

Recommendations

To Administrators

Baseline Teams, Manage Configurations, and Monitor ACL Changes

Treat teams as a critical application and baseline and enforce the ACLs that protect it. Modification of these ACLs to extend read file access outside of the intended user will effectively expose that user's credentials to any of the malicious actions emphasized above.

Once Microsoft has updated the Electron Teams applications, it is still critical to move to a high-restriction model for preventing the installation of unauthorized Teams Apps, bots, connectors, etc.

Track File Access

Create a system monitoring rule to identify the processes accessing these sensitive files. There are two specific file/folder recommendations:

• [Windows] %AppData%\Microsoft\Teams\Cookies

• [Windows] %AppData%\Microsoft\Teams\Local Storage\leveldb

• [macOS] ~/Library/Application Support/Microsoft/Teams/Cookies

• [macOS] ~/Library/Application Support/Microsoft/Teams/Local Storage/leveldb

• [Linux] ~/.config/Microsoft/Microsoft Teams/Cookies

• [Linux] ~/.config/Microsoft/Microsoft Teams/Local Storage/leveldb

If any process other than Teams.exe accesses these files, it indicates that the stored data is being accessed outside the context of the Teams application.

Consider the Web App as an Alternative

If baselining and monitoring is impractical, consider using the web-based Teams client inside Microsoft Edge, which has multiple OS-level controls to protect token leaks. Fortunately, the Teams web application is robust and supports most features enabled through the desktop client, keeping organization productivity impacts to a minimum.

For Linux users, this is the recommended path full stop as Microsoft has announced the end of life for Teams for Linux by December 2022.

To Developers

If you must use Electron for your application, ensure you securely store OAuth tokens. One such method for storing secrets is to use the package KeyTar, which leverages local OS security mechanisms for secret management.

How to securely store sensitive information in Electron with node-keytar | by Cameron Nokes | Cameron Nokes | Medium

To Microsoft

If you must store the tokens, do so in an encrypted fashion and raise the bar from a simple read access on the file system to requiring some ongoing access to memory. And please let us remove disabled accounts from the Teams app(s) without having to do a full uninstall/re-install.

To Security Teams

To stop an attacker that would leverage this type of exploit, teams should invest in threat detection, and response solutions that can see the kinds of actions before and after the exploit are leveraged. Vectra's threat detection and response platform would detect command and control tactics that would be expected before this exploit and any network reconnaissance, lateral movement, or exfiltration that would occur after the exploit. In the cloud, Vectra would alert on the attacker's activities using the compromised credentials in Azure AD, including privileged in Azure AD, full-scale attacks against connected cloud service providers like AWS, and attacks against other Microsoft 365 applications, including Exchange, SharePoint, and Power Automate.

Threat Detection and Response for your Azure AD

To learn more about how Vectra can see and stop threats please visit:
https://www.vectra.ai/products/platform

Want to learn more?

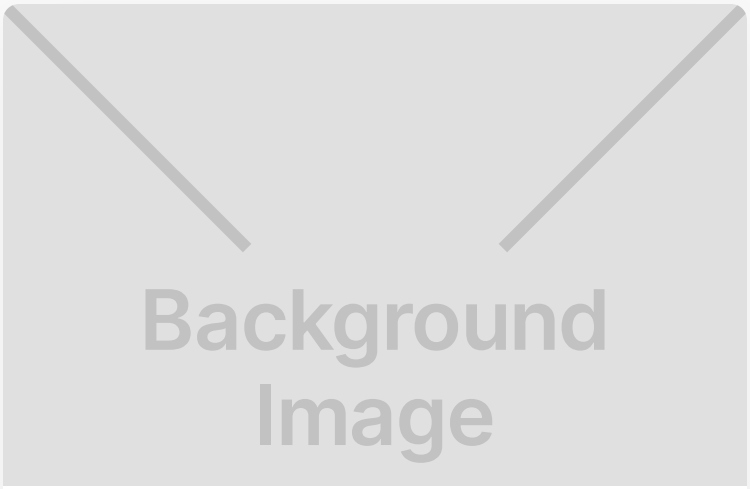
Vectra® is the leader in Security AI-driven hybrid cloud threat detection and response. The Vectra platform and services cover public cloud, SaaS applications, identity systems and network infrastructure – both on-premises and cloud-based. Organizations worldwide rely on the Vectra platform and services for resilience to ransomware, supply chain compromise, identity takeovers, and other cyberattacks impacting their organization.

If you'd like to hear more, contact us and we'll show you exactly how we do this and what you can do to protect your data. We can also put you in contact with one of our customers to hear directly from

them about their experiences with our solution.

Contact us >

Related blogs

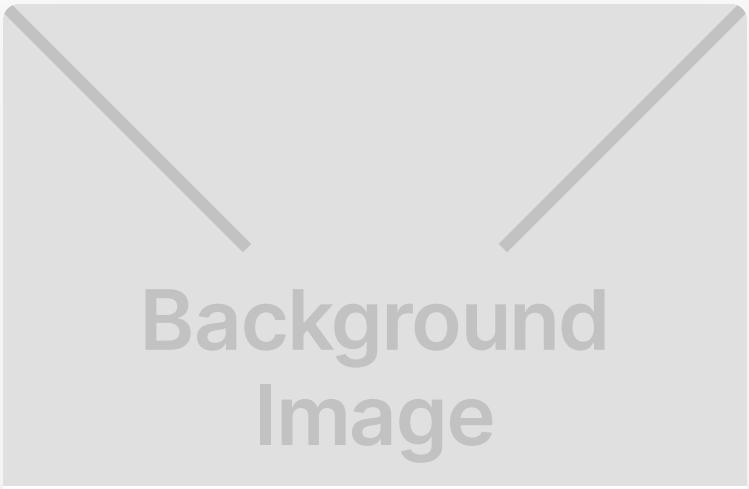


January 20, 2024

Blocking Shodan

Explore the pervasive impact of Shodan on network security, from its origin to its contemporary applications in cybersecurity.

Read more >

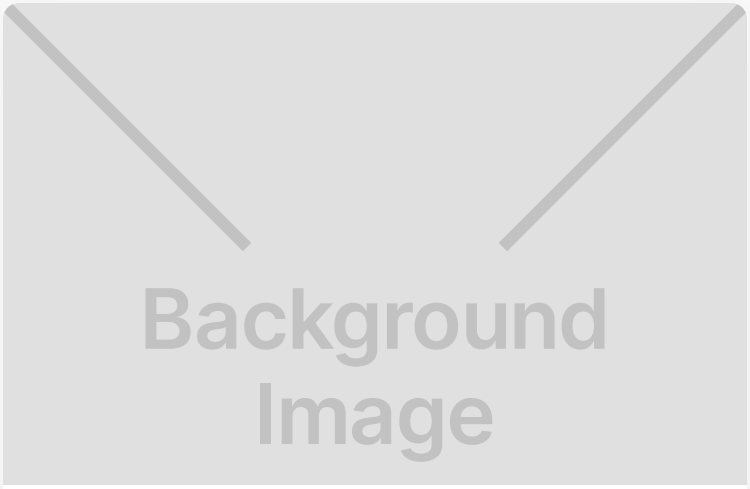


January 25, 2022

Log4J Won't Be the Last Exploit, So Let's Make Sure You're Ready

As we saw with the Log4J vulnerability, cybercriminals only need a single opening to infiltrate your environment. And while another vulnerability can't be prevented, there's still a lot that can be done to make sure you're ready for the next one.

Read more >



December 20, 2021

Log4J's Unique Impact In The Cloud

Threat actors can use the Log4J vulnerability as a platform for launching attacks, but what does this mean for cloud environments? Find out exactly how attackers are exploiting this vulnerability and what this could mean for your organization.

Read more >

VECTRA®

Platform

Public Cloud

SaaS

Identity

Network

Endpoint

Managed Extended Detection & Response Services

See our Integrations ›

Our AI

Vectra AI Detections ›

Product in Action

Vectra AI Platform Video Demo

Vectra AI Platform Tour

Stop a hybrid attack tour

Stop a ransomware tour

Stop an AWS attack tour

Use Cases

SOC MODERNIZATION

EDR Extension

IDS Replacement

PCAP Replacement

SIEM Optimization

Signature + AI-driven Detection

CYBER RESILIENCE

Cloud Identity Protection

Cloud Control Plane Protection

Cloud Posture Improvement

RISK MANAGEMENT

OT Environment Risk

Critical Infrastructure Risk

Remote Workforce Risk

See all Use Cases ›

Hybrid Attack Types

Account Takeover

Advanced Persistent Threats

Data Breach

Nation State Attacks

Ransomware

Supply Chain Attacks

Hybrid Attacks Progressions

Zero-day Exploit

Spear Phishing

MFA Bypass

Credential Stuffing

Sunburst

Living off the Land

Industries

Critical National Infrastructure

Energy & Utilities

Finance

Government/Federal

Healthcare

Higher Education

Manufacturing

Pharmaceutical & medical

Real Estate

Retail & Wholesale

Telecom

Contact Us

support@vectra.ai

Headquarters
550 S. Winchester Blvd.
Suite 200
San Jose, CA, USA 95128