

Adversary-in-the-Middle in Practice

Anatomy of a Microsoft 365 Session Theft, from Relayed MFA to Scripted Graph Reconnaissance

Chris Jordan, Fluency Security · October 14, 2026 · Sanitized case study

ABSTRACT. An adversary-in-the-middle (AiTM) phishing kit captured a fully MFA-satisfied Microsoft 365 session from one employee, and the attacker replayed it 21 hours later to run scripted Microsoft Graph reconnaissance aimed at payroll, HR and finance staff. This paper walks through that intrusion end to end, using sanitized telemetry from a real customer environment ("Customer A", a large US construction firm). It pairs the case with the public research summarized in Fluency's October 2026 AiTM customer brief. The case is useful because it is ordinary: no zero-day and no privileged account. The opening was a legitimate MFA recovery, not helpdesk social engineering. The user's Authenticator app malfunctioned, so the helpdesk added the user's own Android phone number for SMS in order to enroll a new company phone. That Android was very likely compromised, and the SMS code sent to it completed an attacker's relayed sign-in. From there the attacker needed only a residential proxy network and a canned Graph script.

Key findings

1. **MFA was satisfied, not bypassed.** The capture sign-in completed password + SMS one-time code. Every later replay carried the `multiFactorAuthentication` claim, and the attacker never answered another prompt.
2. **The weak moment was an MFA recovery onto a likely compromised phone.** Authenticator on the user's personal Android malfunctioned during a phone migration. At the user's request, the helpdesk added that Android's number for SMS; the code sent to it completed the attacker's sign-in 14 minutes later.
3. **Dwell, then burst.** The session sat idle for about 21 hours, then produced two bursts of Graph activity totalling under 50 minutes.
4. **The post-compromise tell is the Graph query, not the sign-in.** The attacker pulled 999 directory records with an `axios` script, then ran a 60-term `$search` for payroll, HR, finance and admin keywords. That query is a reusable fingerprint of the kit, and the API calls sent a hard-coded Firefox 131 user agent while the sign-ins claimed Firefox 151: the session was driven by a tool.
5. **One session, many identities.** The same session ID appeared from 17 residential IPs across 7+ US states and Canada, with three different browser strings. Session-level correlation found it; per-IP and per-sign-in risk scoring did not (Identity Protection rated every replay "none").
6. **Containment had a hybrid-identity gap.** An on-premises password reset did not reliably take effect in the cloud, and the attacker re-entered a valid password after revocation. Phone-based push approval was the last control that held.

1 How AiTM works

AiTM phishing steals the authenticated session that Microsoft issues after MFA succeeds, so the attacker never needs to defeat MFA itself. The kit runs a reverse proxy between the victim and the real Microsoft sign-in service. The victim sees genuine Microsoft pages, enters a real password and completes a real MFA challenge. The proxy keeps the resulting session cookie and tokens.

The five stages, as described in Fluency's October 2026 customer brief, are shown in Figure 1:

1. **Lure.** An email, SMS, QR code, shared-file notice or search result sends the user to an attacker domain.
2. **Reverse proxy.** The kit relays the genuine Microsoft login in real time.
3. **Authentication.** The victim completes password and MFA against the real service.
4. **Session theft.** The proxy captures the session cookie (ESTSAUTH / ESTSAUTHPERSISTENT) and any tokens issued.
5. **Replay and abuse.** The attacker replays the session from their own infrastructure and requests fresh tokens for Exchange, Graph or SharePoint.

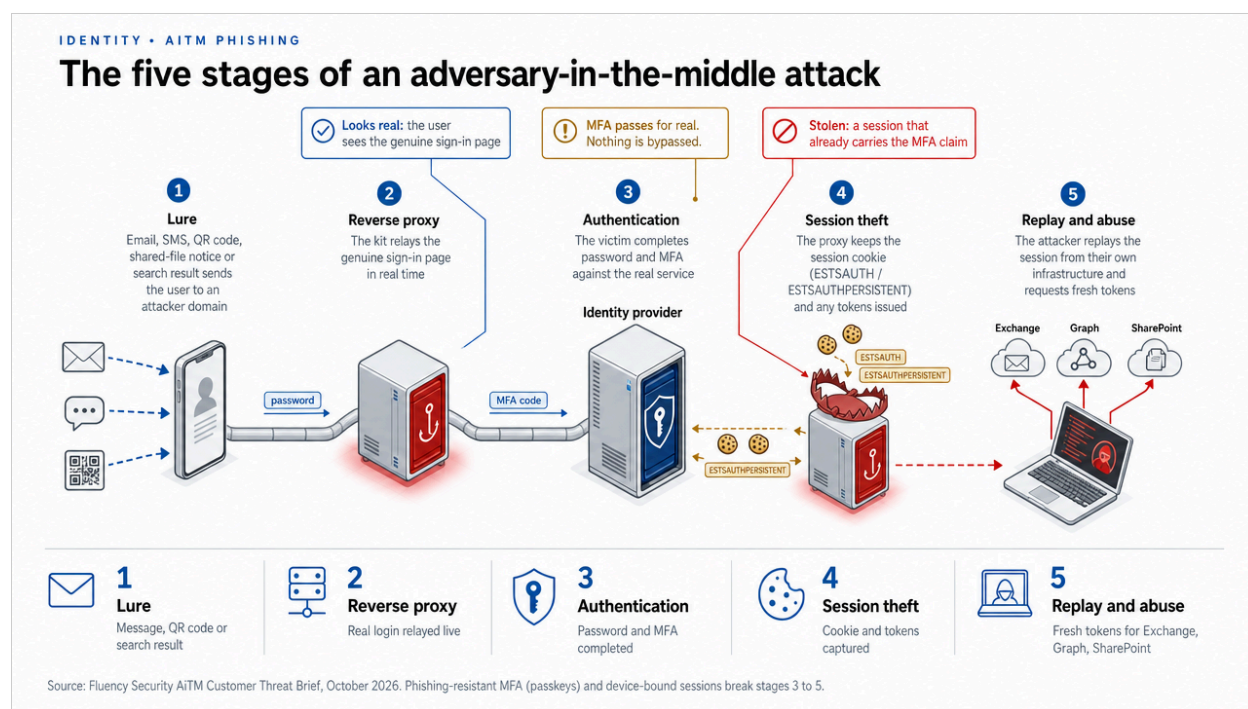


Figure 1. The five stages of an adversary-in-the-middle attack: lure, reverse proxy, authentication, session theft, replay and abuse.

Why commonly deployed MFA does not help. Push, SMS and TOTP codes are all phishable: the victim types or approves them, and the proxy forwards them. Only origin-bound methods (FIDO2 passkeys, Windows Hello for Business, certificate-based auth) fail against a proxy, because the credential is cryptographically tied to the real Microsoft domain.

Table 1. Authentication artefacts involved in AiTM session theft.

Artefact	Where it lives	Typical lifetime	What replay gives the attacker
Session cookie (ESTSAUTH*)	Browser, sent to login.microsoftonline.-com	Hours to days; extended by "Stay signed in"	New sign-ins without password or MFA; shows as a sign-in with an existing SessionId
Refresh token	App or browser storage	Up to 90 days, sliding	New access tokens for any resource the client app is consented to
Access token	Sent with each API call	About 60–90 minutes	Direct calls to Graph, Exchange Online, SharePoint

Scale. The brief notes that the Tycoon2FA phishing-as-a-service platform reached more than 500,000 organisations a month before its 2026 disruption, and that Microsoft has observed payment fraud starting as little as five minutes after session theft. AiTM is a commodity capability, not a bespoke one.

ATT&CK classifies the capture as T1557 (Adversary-in-the-Middle) and T1539 (Steal Web Session Cookie), and the replay as T1550.004 (Use Alternate Authentication Material: Web Session Cookie).

2 Case study: Customer A

One non-privileged user was compromised; the attacker held a working session for about 22 hours and actively used it for under 50 minutes. Customer A is a large US general contractor with a hybrid identity setup: on-premises Active Directory synced to Microsoft Entra ID, Exchange Online, Defender XDR, Entra ID Protection, Jamf-managed iPhones and an email security gateway. Fluency ingested Entra sign-in logs (interactive and non-interactive), Microsoft Graph activity logs, the Office 365 unified audit log, Defender for Identity, Defender email and URL-click events, and Jamf Security events.

Table 2. Case summary.

Item	Detail
Victim	"User A", field operations role, no directory roles
Victim devices	Personal Android phone (unmanaged, not Entra-registered; very likely compromised); new company-issued iPhone (Jamf-enrolled during the incident)
Initial MFA methods	Authenticator push on the Android (malfunctioning); SMS to the Android's number, added by the helpdesk at the user's request
Capture	Day 0, 2:14 PM local
First replay	Day 1, 11:02 AM local (about 21 hours later)
Active abuse	Day 1, 11:04–11:55 AM local
First alert	Defender XDR, Day 1, 11:55 AM, High, MITRE T1591 (Gather Victim Org Information)
Containment	Sessions revoked Day 1, 12:15 PM; on-prem reset and disable 12:21 PM
Last attacker attempt	Day 1, 7:04 PM (failed)

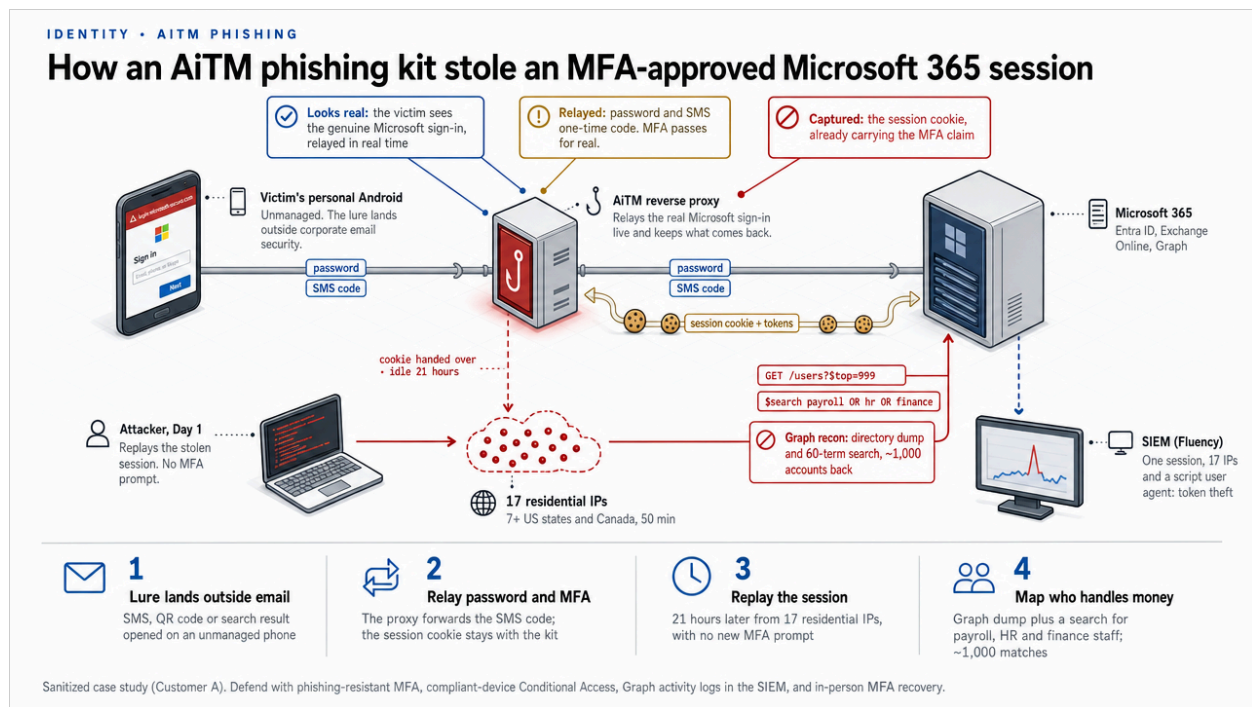


Figure 2. The Customer A attack chain: relayed capture on an unmanaged phone, 21-hour dwell, replay through 17 residential IPs, and Graph reconnaissance. The attacker's active time was under 50 minutes; the 21-hour gap between capture and use is where session-level detection has the most room to act.

Sanitization. Names, tenant identifiers, phone numbers, ticket numbers, session and token IDs and full IP addresses are removed or truncated. Times are local (US Central) and expressed as Day 0 / Day 1 relative to the capture. IP addresses are described by network type and approximate geolocation only. User-agent strings, Graph request paths, error codes and the attacker's `$search` query are reproduced exactly: they describe the attacker's tooling, not the victim, and they are the most reusable detection material in this paper.

3 Phase 1: Initial access (Day 0)

The session was captured at 2:14:11 PM on Day 0, 14 minutes after the helpdesk, at the user's request, added the user's personal Android number as an SMS method so a new company phone could be enrolled.

Precondition: a broken MFA method. User A was moving to a new company iPhone. From 1:21 to 1:40 PM, the logs show a loop the user could not escape. The Jamf enrollment on the iPhone needed push MFA. Authenticator on the old Android needed MFA to re-add the account. The only method available was the push that was not arriving. The responding team reports that Authenticator on the Android had been behaving erratically, which is consistent with the device being compromised. The new iPhone's own number was already listed for SMS, but it could not serve as the MFA method while that same phone was being set up, so the helpdesk added the Android's number instead.

Table 3. Day 0 sign-in sequence leading to capture.

Time	Device	Event	Result code
1:21 PM	iPhone, home ISP	Jamf enrollment sign-in, password correct	Push denied

Time	Device	Event	Result code
1:22 PM	Android	Saved Microsoft session expired for inactivity	700082
1:23–1:34 PM	Android, Authenticator	7 attempts to re-add the account, each needing push MFA	Push in progress / failed
1:30–1:40 PM	iPhone, home ISP	Jamf enrollment retried 3 times	Push denied / never completed
1:31–1:36 PM	Android, Chrome	My Sign-ins (security info) and M365 support pages	MFA required, push fails
2:00:44 PM	Helpdesk technician, corporate network	At the user's request, adds the user's Android number as an SMS method in the Entra portal	Audit: user registered security info
2:02 PM	iPhone, corporate network	Jamf enrollment succeeds	Success
2:12:04 PM	Android, Chrome	Opens Outlook on the web	MFA required
2:14:11 PM	Attacker proxy, residential ISP, North Carolina	OfficeHome sign-in: password + the SMS code sent to the Android. New SessionId created.	Success
2:14:16 PM	Android, Chrome	Lands on the real Office home page under its own session	Push pending
2:15:32 PM	iPhone, corporate network	User A completes SMS MFA to the same number	Success

Tells in the capture sign-in

- **Browser/OS mismatch.** The user agent was the Android Chrome string `Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/154.0.0.0 Mobile Safari/537.36`, but Entra's device detail parsed the OS as Windows 10. The proxy forwards the victim's UA header while its own TLS and client hints come from a Windows host. Caveat: Chrome's reduced Android UA is shared by every Chrome 154 Android user, so a matching string is consistent with the victim but does not prove it.
- **Gap-and-redirect timing.** The capture falls inside a two-minute gap in the Android's Chrome activity (2:12:04 to 2:14:16). Five seconds after capture, the Android lands on the genuine Office home page. Kits redirect the victim to the real site once they have the session, so nothing looks wrong.
- **Geography.** The capture IP was residential IPv6 in North Carolina; the user was in Missouri on a cellular network.
- **No email lure.** Defender email and Safe Links click logs showed no relevant click. The compromise most likely ran through the unmanaged Android itself, outside every corporate telemetry source: either malware already on the device or a lure opened on it (SMS, QR code, search result, personal email). Jamf's network-traffic feed, which might have seen it on a managed phone, covered only the iPhone and had a 7-day ingestion gap.

Root cause hypotheses. Three explanations were tested against the logs and the responding team's account (Table 4).

Table 4. Root cause hypotheses.

Hypothesis	Evidence	Verdict
Helpdesk social engineering: an attacker posing as User A had their own number added	The added number belongs to User A's own Android; the helpdesk acted on the real user's request; User A completed SMS to that number at 2:15 PM and again on Day 2	Ruled out
User A's Android was compromised, and the SMS code sent to it reached the attacker along with the password	Responding team's assessment; Authenticator on the Android behaved erratically beforehand; the attacker used an SMS code to that number within 14 minutes of it being added	Likely
An AiTM phishing page was opened on the Android and the user entered the code	Browser/OS mismatch; 2-minute gap in the Android's Chrome activity; redirect to the real Office page 5 s after capture; no phishing URL recovered	Possible; may be how the device compromise played out

Either way, the sign-in itself was relayed from attacker infrastructure, which is why this paper treats the case as AiTM session theft. The lesson: an MFA recovery that registers a phishable method on a device nobody has checked can hand the attacker the second factor. Device health belongs on the recovery checklist.

4 Phase 2: Dwell and session replay (Day 1)

The attacker left the session untouched for 20 hours 48 minutes, then replayed it through a rotating residential proxy that changed IP on nearly every request. A long dwell is common in kit-as-a-service operations: one crew captures sessions, another works them later, often during the victim's business hours to blend in.

Table 5. Replay chain. Every row carries the same SessionId as the Day 0 capture.

Time	Sign-in type	Network (sanitized)	Resource	Result
11:02:54	Interactive, existing session	Residential, Florida	Exchange Online	50199 (confirmation interrupt)
11:04:02	Interactive, "MFA skipped, remembered device"	Residential, Montana	Exchange Online via Microsoft Outlook client	Success; refresh token issued
11:04:07	Non-interactive (refresh)	Residential	Microsoft Graph	Access token used for <code>/me</code> and <code>/users</code>
11:04:20	Non-interactive	Residential	Exchange Online	Success
11:17	Non-interactive	Residential, British Columbia	Exchange Online	Success
11:19	Non-interactive	Residential, New Mexico	Microsoft Graph	Success
11:44:37	Non-interactive	Residential	Exchange Online	New access token, used for mailbox reads
11:44:39	Non-interactive	Residential	Microsoft Graph	New access token, used for <code>\$search</code>
11:51–11:53	Interactive	Residential, North Carolina / California	Exchange Online	50199

Time	Sign-in type	Network (sanitized)	Resource	Result
11:55	Non-interactive	Residential, Arizona	Microsoft Graph	Success (last successful use)

What this phase tells a defender

- **Session-level travel, not user-level travel.** Seventeen distinct IPs across at least seven US states and Canada in under an hour, all on residential ISPs (Charter, Frontier, AT&T and other consumer ASNs). Per-IP reputation is useless here; each IP is a clean home connection. The impossibility only appears when events are grouped by SessionId.
- **MFA claim inherited.** Every replay showed `authenticationRequirement = multiFactorAuthentication` satisfied by the original Day 0 claim. Nothing asked the attacker for a second factor.
- **Client switch.** The capture was an OfficeHome browser sign-in. The replay requested tokens for the **Microsoft Outlook** client app, whose pre-consented scopes include `Mail.ReadWrite`, `Chat.ReadWrite`, `Files.ReadWrite.All`, `Group.ReadWrite.All`, `User.Invite.All` and `User.ReadBasic.All`. Switching to a first-party client gives a broad, already-consented Graph token with no OAuth consent event to detect.
- **Unmanaged, non-compliant device.** Every replay came from an unregistered, non-compliant device. Conditional Access did not require a compliant device for Exchange or Graph, so this was not blocking.
- **Identity Protection was silent.** Sign-in risk was "none" on every replay. Identity Protection raised a High "threat intelligence" user risk only at 12:45 PM, 30 minutes after an admin had already revoked the sessions.
- **Clock skew is expected.** Graph activity logs record token issue times about 5 minutes earlier than the sign-in log, because Entra backdates `iat` to tolerate skew. Correlate on the unique token identifier, not the timestamp.

5 Phase 3: Post-compromise Graph reconnaissance

The attacker made five API calls in two bursts: confirm identity, dump the directory, confirm identity again, search the directory for payroll/HR/finance people, then test mailbox access. No mail was sent, no rules were created, and Teams, OneDrive and SharePoint were not touched, even though the token allowed all of them.

Table 6. Attacker API activity, Day 1.

Time	Network (sanitized)	User agent	Request	Response
11:04:12	Residential	Firefox 131	GET /v1.0/me	200
11:04:16	Residential	axios/1.18.1	GET /v1.0/users?\$top=999	200, 316 KB
11:44:42	Residential (new IP)	Firefox 131	GET /v1.0/me	200
11:44:49	Residential (new IP; the alerted address)	Firefox 131	GET /v1.0/users?\$search=...&\$top=999	200, 229 KB
11:45:06	Residential, California	Firefox 131	Exchange: 3 messages opened (MailItems-Accessed, Bind)	Success

The targeted \$search

URL-decoded, the 11:44:49 request was:

```
GET https://graph.microsoft.com/v1.0/users
?$search=
  "displayName:payroll" OR "givenName:payroll" OR "surname:payroll" OR "jobTitle:payroll" OR
"mail:payroll" OR "userPrincipalName:payroll"
OR "displayName:pay" OR "givenName:pay" OR "surname:pay" OR "jobTitle:pay" OR
"mail:pay" OR "userPrincipalName:pay"
OR "displayName:hr" OR "givenName:hr" OR "surname:hr" OR "jobTitle:hr" OR
"mail:hr" OR "userPrincipalName:hr"
OR "displayName:human" OR "givenName:human" OR "surname:human" OR "jobTitle:human" OR
"mail:human" OR "userPrincipalName:human"
OR "displayName:resources" OR "givenName:resources" OR "surname:resources" OR "jobTitle:resources" OR
"mail:resources" OR "userPrincipalName:resources"
OR "displayName:support" OR "givenName:support" OR "surname:support" OR "jobTitle:support" OR
"mail:support" OR "userPrincipalName:support"
OR "displayName:info" OR "givenName:info" OR "surname:info" OR "jobTitle:info" OR
"mail:info" OR "userPrincipalName:info"
OR "displayName:finance" OR "givenName:finance" OR "surname:finance" OR "jobTitle:finance" OR
"mail:finance" OR "userPrincipalName:finance"
OR "displayName:account" OR "givenName:account" OR "surname:account" OR "jobTitle:account" OR
"mail:account" OR "userPrincipalName:account"
OR "displayName:admin" OR "givenName:admin" OR "surname:admin" OR "jobTitle:admin" OR
"mail:admin" OR "userPrincipalName:admin"
&$top=999
```

Ten keywords across six attributes, 60 terms joined with OR. Three things mark it as tooling rather than a human:

- **Exhaustive cross-product.** Every keyword is searched in every field, including meaningless pairs such as `surname:admin` and `givenName:resources`. A person would not type this; a template would.
- **Prefix matching widens the net.** Graph `$search` on users tokenizes and matches word prefixes, so `pay` also hits surnames like "Payne" and titles like "Payables", and `admin` hits every "Administrative Assistant". Re-running the query against Customer A's directory returned **1,016** accounts: 121 enabled payroll, accounts-payable, finance and HR staff; about 334 guest accounts (subcontractors and vendors, some matched on company name); shared mailboxes such as `payroll@`; and a long tail of noise. The 999-row cap means the attacker received almost all of them.
- **Response size fits default fields.** 229,276 bytes for 999 records is about 230 bytes each, consistent with Graph's default user projection, which includes `mobilePhone` and `businessPhones`. Assume the attacker has phone numbers for every match, which enables SMS and voice follow-on (smishing, vishing).

Intent. The keyword set maps directly to payroll diversion and business email compromise: find who can change direct deposit, who pays vendors, who resets accounts, and which shared mailboxes to spoof. MITRE: T1087.004 (Cloud Account Discovery) and T1591.004 (Identify Roles).

The user-agent tell. Two strings were used, and neither matched the sign-ins:

- `axios/1.18.1` on the full directory dump. Axios is a Node.js HTTP library; a browser never sends it.
- `Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:131.0) Gecko/20100101 Firefox/131.0` on every other call. The interactive replays claimed Firefox 151 and 153. A browser does not change ma-

for version inside one session, so Firefox 131 is almost certainly hard-coded in the kit's API client.

A UA that differs between the sign-in that minted a token and the API calls that spend it is one of the cleanest token-theft signals available, because a legitimate client rarely does this (see the baseline note below).

5.1 User-agent analysis: one session, four browsers

The stolen session presented four different browser identities in about 26 hours, and the tokens tie them together: every Graph and Exchange call carried the unique token ID of a token minted under the Day 1 "Firefox 151" sign-in, yet none of those calls sent Firefox 151. That split between the identity shown at sign-in and the identity shown at the API is the signature of an automated tool, not a person in a browser.

Table 7. Browser identities presented by the stolen session.

Stage	Time	Event	User agent presented
Capture	D0 2:14:11 PM	Proxied OfficeHome sign-in	Chrome Mobile 154 (Android), OS parsed as Windows 10
Replay	D1 11:02:54, 11:04:02 AM	Interactive replay; Outlook refresh token issued	Firefox 151 on Windows
Replay	D1, other interactive replays	Interactive replay attempts	Firefox 153 on Windows
API	D1 11:04:16 AM	GET /users?\$top=999	axios/1.18.1
API	D1 11:04:12 – 11:45:06 AM	/me , \$search , mailbox reads	Firefox 131 on Windows
Return	D1 3:45–3:46 PM	Replay and password sign-in	Chrome 153 on Windows

Token-level join. Each API call in Table 8 was authorized by a token whose ID appears in the sign-in logs; all of these tokens descend from the refresh token issued at the 11:04:02 Firefox 151 sign-in. Token IDs are replaced with labels.

Table 8. Tokens minted under "Firefox 151", spent by a different client.

Token	Issued (sign-in log)	Resource	Spent (activity log)	UA at the API	Request
G1	11:04:07	Microsoft Graph	11:04:12	Firefox 131	GET /me
G1	11:04:07	Microsoft Graph	11:04:16	axios/1.18.1	GET /users?\$top=999
G2	11:44:39	Microsoft Graph	11:44:42	Firefox 131	GET /me
G2	11:44:39	Microsoft Graph	11:44:49	Firefox 131	60-term \$search
E2	11:44:37	Exchange Online	11:45:06	Firefox 131	3 messages opened

Why this proves tooling.

- **One token, two clients, four seconds.** Token G1 was spent by "Firefox 131" at 11:04:12 and by axios at 11:04:16. A browser cannot hand its token to a Node.js library mid-session; a script that sets its UA per request can.
- **The version never moves.** Firefox ships a major release about every four weeks, so 131 is about 20 releases, roughly a year and a half, older than the Firefox 151 the same session

claimed at sign-in. Every API call across both bursts sent exactly `rv:131.0`. A frozen, stale string is a constant in code, not a browser that auto-updates.

- **Sign-in and API identities are set separately.** The kit forwards or spoofs a current browser string at sign-in, where Conditional Access and risk scoring look, and uses its own hard-coded client string at the API, where fewer controls look. The sign-in UA also drifted (Firefox 151 and 153 on Day 1 replays, Chrome 153 on the afternoon return), consistent with an operator rotating presets.
- **Baseline before alerting.** A legitimate Outlook or browser client normally presents the same UA family at sign-in and at the API for the life of a token, but mobile apps and Office clients can differ by design. Baseline your own tenant's sign-in-to-API UA pairs before treating every mismatch as malicious; a script library or a frozen, stale browser version is the high-confidence case.

Detection T4 (Section 7) automates this join. A stricter variant flags any Graph call whose Firefox or Chrome major version is more than 6 releases behind the version on the sign-in that issued its token.

Mailbox access. At 11:45:06 the attacker opened the top three Inbox items (a newsletter, a Viva Engage digest and an internal announcement) within the same second. That is an access check, not a search. The MailItemsAccessed record had a blank user object ID and was found only by joining on SessionId; a hunt keyed on user ID alone would have missed it.

6 Phase 4: Return attempts after containment

The attacker came back four times after the sessions were revoked and, on at least two attempts, entered a valid password; push MFA on the user's phone was the only control that stopped them. This phase is where hybrid-identity environments most often fail.

Table 9. Containment actions and return attempts, Day 1.

Time	Actor	Event	Result
12:15 PM	Admin	Revoke all sessions in Entra	Refresh tokens invalidated
12:21 PM	Server admin	Password reset + "must change at next logon" + disable, on an on-prem domain controller	Recorded by Defender for Identity only
12:45 PM	Identity Protection	High user risk (threat intelligence); tokens revoked again automatically	—
1:36 PM	Server admin	Account re-enabled on-prem (synced to Entra 1:57 PM)	—
3:45:35 PM	Attacker, hosting ASN, New Jersey, Chrome 153 / Windows	Replay of stolen SessionId	70046 (session revoked)
3:46:03 PM	Same	Fresh interactive sign-in, password correct	50074 (push pending, never approved)
4:41 PM	Attacker, residential, Kansas City area	Password correct	Push never approved
7:04 PM	Attacker, residential, New Mexico	Token replay	Rejected

Why the password still worked. Two explanations fit, and the second was supported by later review:

1. The temporary password was disclosed (phished again through the same kit, or given to someone impersonating the user).
2. The on-prem reset never took effect in Entra. A reset synced from on-prem AD with password hash sync creates no Entra audit event, and "must change at next logon" handling with PHS can leave the cloud accepting the previous hash. A later review concluded the original captured password was most likely still valid in the cloud.

Either way, a password reset done only on-prem is not proof of credential rotation in the cloud. Confirm with the Entra sign-in log (a failed attempt with the old password, error 50126) or with `lastPasswordChangeDateTime` on the cloud user object.

What revocation did and did not do. Revoking sessions killed the stolen session cookie and refresh tokens (70046 on replay). It did not stop the attacker starting a fresh sign-in with a password they still had. Revocation also does not instantly kill access tokens already issued, which remain valid for up to their lifetime (roughly an hour) unless the resource supports Continuous Access Evaluation. Exchange Online and Graph do, which is one reason the 12:15 PM revocation was effective.

7 Detection tells and hunting queries

No single event in this intrusion looked malicious on its own; every useful detection came from joining sign-ins, Graph activity and audit records on SessionId or token ID. Table 10 ranks the tells by how early they fire and how specific they are. The queries below are written for Microsoft Sentinel / Log Analytics and should be validated against the reader's own workspace schema before deployment.

Table 10. Detection tells.

#	Tell	Source	Fires at	Specificity
T1	One SessionId seen from several ASNs or countries within an hour	Sign-in logs (interactive + non-interactive)	First replay	High
T2	Browser string parsed as mobile but OS parsed as desktop on an interactive sign-in	SigninLogs	Capture	Medium
T3	New phishable MFA method (SMS/voice) followed by a successful sign-in from a new ASN within 60 minutes	AuditLogs + SigninLogs	Capture	High
T4	Graph call user agent differs from the UA of the sign-in that issued the token, or is a script library (<code>axios</code> , <code>python-requests</code> , <code>node-fetch</code> , <code>Go-http-client</code>)	MicrosoftGraphActivityLogs	First API call	High
T5	<code>/users?\$top=999</code> or <code>\$search</code> across <code>jobTitle/displayName</code> with payroll, HR, finance or admin keywords, from a non-admin	MicrosoftGraphActivityLogs	Reconnaissance	Very high
T6	Interactive sign-in with valid password after a password reset or session revocation	SigninLogs + IdentityDirectoryEvents	Return	High

T1: Session spread across networks.

```

union SigninLogs, AADNonInteractiveUserSignInLogs
| where TimeGenerated > ago(1d) and ResultType == 0
| summarize IPs = dcount(IPAddress), ASNs = dcount(AutonomousSystemNumber),
    Regions = make_set(tostring(LocationDetails.state), 20),
    First = min(TimeGenerated), Last = max(TimeGenerated)
    by UserPrincipalName, SessionId
| where ASNs >= 3 and Last - First < 2h
| order by ASNs desc

```

T3: Phishable MFA method added, then a sign-in from somewhere new.

```

let added = AuditLogs
| where OperationName has_any ("User registered security info", "Admin registered security info", "Update user")
| where tostring(TargetResources) has "phone"
| project AddedAt = TimeGenerated, UPN = tostring(TargetResources[0].userPrincipalName),
    AddedBy = tostring(InitiatedBy.user.userPrincipalName), AddedFromIP = tostring(InitiatedBy.user.ipAddress);
added
| join kind=inner (SigninLogs | where ResultType == 0) on $left.UPN == $right.UserPrincipalName
| where TimeGenerated between (AddedAt .. AddedAt + 60m) and IPAddress != AddedFromIP
| where AuthenticationDetails has_any ("Text message", "OneWaySMS")
| project AddedAt, TimeGenerated, UPN, AddedBy, IPAddress, AutonomousSystemNumber, UserAgent, SessionId

```

T4: Token minted by one client, spent by another. `SignInActivityId` in the Graph log equals `UniqueTokenIdentifier` in the sign-in log.

```

let tokens = union SigninLogs, AADNonInteractiveUserSignInLogs
| project UniqueTokenIdentifier, SignInUA = UserAgent, SessionId, UserPrincipalName;
MicrosoftGraphActivityLogs
| where TimeGenerated > ago(1d)
| join kind=inner tokens on $left.SignInActivityId == $right.UniqueTokenIdentifier
| extend ScriptUA = UserAgent matches regex @"(?i)^(axios|python-requests|node-fetch|go-http-client|curl|okhttp)/"
| where ScriptUA or UserAgent != SignInUA
| project TimeGenerated, UserPrincipalName, SessionId, IPAddress, RequestMethod, RequestUri, UserAgent,
    SignInUA, ResponseSizeBytes

```

T5: Directory harvesting for finance roles. The kit's query is a near-exact signature; the broader version catches variants.

```

MicrosoftGraphActivityLogs
| where TimeGenerated > ago(7d) and RequestMethod == "GET"
| extend Uri = url_decode(RequestUri)
| where Uri has "/users"
| extend Bulk = Uri has_any ("$top=999", "top=999"),
    RoleSearch = Uri has "$search" and Uri has_any ("payroll", "finance", "human", "jobTitle:")
| where Bulk or RoleSearch
| summarize Calls = count(), Bytes = sum(ResponseSizeBytes), UAs = make_set(UserAgent),
    IPs = make_set(IPAddress) by UserId, AppId, bin(TimeGenerated, 1h)

```

Expect benign hits from sync tools and admin scripts. They are app-only (no `UserId`) or run from known IPs, which makes them easy to baseline. In Customer A, two app-only integrations generated thousands of `MailItemsAccessed` events per day and were excluded this way.

T6: Valid password after reset.

```

let resets = IdentityDirectoryEvents
| where ActionType == "Account Password changed"
| project ResetAt = Timestamp, UPN = tolower(TargetAccountUpn);
SigninLogs
| where ResultType in ("0", "50074", "50076", "500121")
| extend UPN = tolower(UserPrincipalName)
| join kind=inner resets on UPN
| where TimeGenerated between (ResetAt .. ResetAt + 24h)
| where AutonomousSystemNumber !in (dynamic([/* corporate and known home ASNs */]))
| project ResetAt, TimeGenerated, UPN, ResultType, IPAddress, AutonomousSystemNumber, UserAgent

```

A 50074 (MFA required) from an unfamiliar network shortly after a reset means the password check passed. Treat it as a live credential exposure, not a failed sign-in.

What correlation added. The investigation ran against Fluency's data lake, where grouping the replay events by session rather than by IP turned eight ordinary residential sign-ins into one impossible-travel cluster. Having every source in one place also kept the Day 0 capture, Graph activity and on-prem reset in one timeline, so analysts could walk back from the Day 1 alert to the Day 0 root cause in one investigation rather than across four consoles.

8 Defensive lessons

The controls that would have changed this outcome are, in order of impact: phishing-resistant MFA, a hardened recovery path, device-bound sessions, and response playbooks that verify cloud credential rotation.

- 1. Make the strongest method the only method for recovery.** The intrusion began when a malfunctioning Authenticator was replaced with SMS to the same, likely compromised, phone. Use a Temporary Access Pass issued in person or after verified video ID, and have the user enroll a passkey or Authenticator with number matching during that same session. Never leave SMS as a standing fallback, and treat an authenticator that suddenly misbehaves as a possible sign the device is compromised: don't register a recovery method on that device.
- 2. Move to phishing-resistant MFA.** FIDO2 passkeys (including Authenticator passkeys on the company phone) and Windows Hello fail against a reverse proxy because the credential is bound to the real origin. Enforce them with a Conditional Access authentication-strength policy, starting with finance, HR, payroll and IT support staff, who are the kit's stated targets.
- 3. Bind sessions to managed devices.** Require a compliant or Entra-joined device for Exchange Online, SharePoint and Graph from browsers and rich clients. Every replay here came from an unregistered, non-compliant device. Where available, enable Token Protection for sign-in sessions.
- 4. Shorten what a stolen session is worth.** Set sign-in frequency for unmanaged devices, disable persistent browser sessions on them, and confirm Continuous Access Evaluation is on in strict mode for Exchange and Graph.
- 5. Ingest the logs that carry the evidence.** This case could not have been reconstructed without non-interactive sign-ins, `MicrosoftGraphActivityLogs` and Defender for Identity. Graph activity logs are off by default and must be enabled through Entra diagnostic settings.
- 6. Write a token-theft playbook, not a password playbook.** The order is: revoke sessions in Entra; reset the password **in the cloud** (or confirm on-prem sync took effect); remove any MFA method added in the past 7 days that cannot be verified in person; re-register the user's MFA in

person; then hunt for inbox rules, forwarding, OAuth grants and new devices. Verify the reset by checking `lastPasswordChangeDateTime` in Entra, not the on-prem event.

7. **Warn the people the attacker looked up.** The reconnaissance shows who the next lure will target. Within hours, notify payroll, AP, finance and HR staff and the vendors whose contacts were exposed; require call-back verification on any bank-detail change.
8. **Cover the unmanaged phone.** The compromise ran through a personal Android outside email security, Safe Links and mobile threat defense. Either bring such phones under app protection policies (MAM) with Conditional Access, or block work access from them.

Appendix A Sanitized timeline

Time (local)	Phase	Event	Source
D0 10:23 AM	Context	New-phone setup ticket opened	Helpdesk
D0 1:21–1:40 PM	Precondition	Push MFA failing on iPhone enrollment and Android Authenticator re-add (10+ attempts)	SigninLogs
D0 2:00:44 PM	Precondition	Helpdesk adds the user's Android number for SMS, at the user's request	AuditLogs
D0 2:02 PM	Context	New iPhone enrolls in Jamf	SigninLogs, Jamf
D0 2:14:11 PM	Capture	Proxied OfficeHome sign-in, password + SMS; stolen SessionId created	SigninLogs
D0 2:14:16 PM	Capture	Victim's Android redirected to real Office home	SigninLogs
D0 2:15:43 PM	Context	iPhone registered in Entra	AuditLogs
D1 11:02:54 AM	Replay	First replay of stolen session (50199)	SigninLogs
D1 11:04:02 AM	Replay	Outlook client refresh token issued	SigninLogs
D1 11:04:12 AM	Recon	GET /me	GraphActivityLogs
D1 11:04:16 AM	Recon	GET /users?\$top=999 , axios , 316 KB	GraphActivityLogs
D1 11:17–11:20 AM	Replay	Non-interactive refreshes from BC, NM, AL	AADNonInteractive
D1 11:44:42 AM	Recon	GET /me with new token	GraphActivityLogs
D1 11:44:49 AM	Recon	60-term payroll/HR \$search , 229 KB	GraphActivityLogs
D1 11:45:06 AM	Collection	3 Inbox messages opened	Unified audit log
D1 11:55 AM	Detection	Defender XDR incident, High, T1591	Defender XDR
D1 11:55 AM	Replay	Last successful token use	AADNonInteractive
D1 12:15 PM	Response	Sessions revoked	AuditLogs
D1 12:21 PM	Response	On-prem password reset and disable	Defender for Identity
D1 12:45 PM	Detection	Identity Protection High user risk; auto-revoke	Identity Protection
D1 1:36 PM	Response	Account re-enabled on-prem	Defender for Identity
D1 3:45 PM	Return	Session replay rejected (70046)	SigninLogs
D1 3:46 PM	Return	Valid password; push pending (50074)	SigninLogs
D1 4:41 PM	Return	Valid password; push never approved	SigninLogs
D1 7:04 PM	Return	Token replay rejected	SigninLogs
D2 6:46–6:48 AM	Response	User blocked by risk policy; self-service password reset	SigninLogs

Time (local)	Phase	Event	Source
D3	Response	In-person MFA rebuild; SMS and personal phone removed	Helpdesk

Appendix B Indicator patterns

These describe the kit and its operator, not the victim. Network indicators are not listed because every attacker IP was a short-lived residential proxy exit; blocking them has little value.

Type	Value	Notes
API user agent	axios/1.18.1	Bulk /users dump
API user agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:131.0) Gecko/20100101 Firefox/131.0	Hard-coded; spent tokens minted under a Firefox 151 sign-in, about 20 releases newer
Sign-in user agents	Firefox 151 and 153 on Windows; Chrome 153 on Windows	Interactive replays and return attempts
Capture fingerprint	Chrome Mobile UA with OS parsed as Windows 10	Proxied mobile sign-in
Client app on replay	Microsoft Outlook (first-party), resource Exchange Online and Microsoft Graph	No consent event
Graph request	GET /v1.0/users?\$top=999	Directory dump
Graph request	\$search over displayName, givenName, surname, jobTitle, mail, userPrincipalName for payroll, pay, hr, human, resources, support, info, finance, account, admin	Kit's role-targeting template
Infrastructure	Residential ISP exits (Charter, Frontier, AT&T and others), one hosting ASN (AS19318)	17 IPs, 7+ states and Canada
Timing	~21 h dwell; bursts of 1 min and 2 min; mailbox check within 20 s of \$search	Scripted operation

Appendix C MITRE ATT&CK mapping

Tactic	Technique	Evidence in this case
Initial Access	Undetermined: compromised personal Android (likely) or T1566.002 Spearphishing Link opened on it	No corporate lure found; the Android was not forensically examined
Credential Access	T1557 Adversary-in-the-Middle	Proxied sign-in with UA/OS mismatch
Credential Access	T1539 Steal Web Session Cookie	SessionId reused from new infrastructure 21 h later
Credential Access	T1111 Multi-Factor Authentication Interception	Relayed SMS one-time code
Defense Evasion / Lateral Movement	T1550.004 Use Alternate Authentication Material: Web Session Cookie	Replays inheriting the MFA claim
Defense Evasion	T1090.002 Proxy: External Proxy	17 residential exits
Discovery	T1087.004 Account Discovery: Cloud Account	/users?\$top=999

Tactic	Technique	Evidence in this case
Reconnaissance	T1591.004 Gather Victim Org Information: Identify Roles	Payroll/HR/finance \$search
Collection	T1114.002 Email Collection: Remote Email Collection	3 Inbox messages opened
Credential Access	T1078.004 Valid Accounts: Cloud Accounts	Valid password entered after reset

References

1. Fluency Security, *Adversary-in-the-Middle (AiTM) Attacks: Customer Threat Brief*, October 2026 (internal; cites Microsoft Security blog posts of 12 July 2022, 21 January 2026 and 4 March 2026, the Microsoft Learn token-theft playbook, and MITRE ATT&CK T1539).
2. Customer A incident investigation, Fluency data lake: Entra sign-in logs, non-interactive sign-in logs, Microsoft Graph activity logs, Office 365 unified audit log, Defender XDR, Defender for Identity, Jamf Security.

Figures 1 and 2 were generated with ElevenLabs image generation from the case data described in this paper.