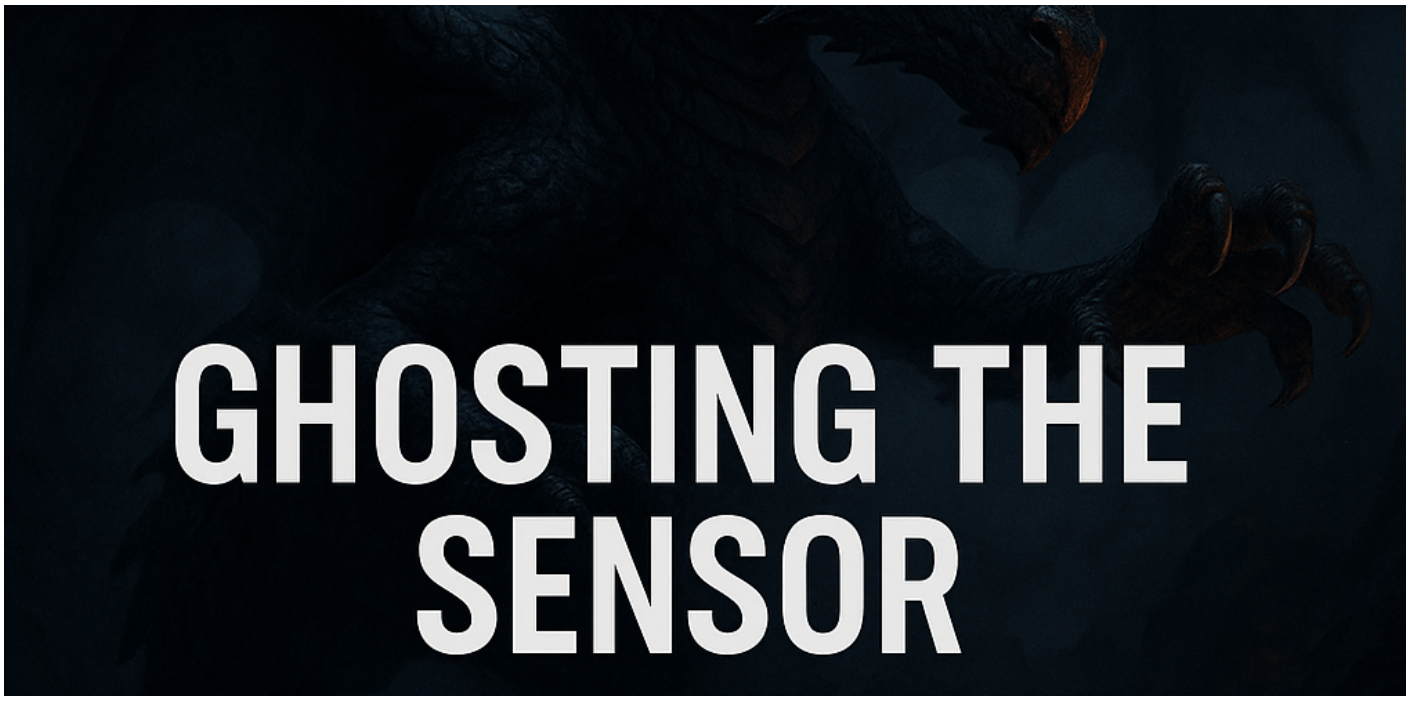
Ghosting the Sensor: Disrupting Defender for Identity Without Detection

⋮ 7/26/2025



Think of your enterprise as a fortified city, and each domain controller as a gatehouse. Microsoft Defender for Identity is like a watchtower built inside the gate, quietly observing everyone who passes through. It doesn't rely on patrols or wall guards but monitors credentials, behaviors, and hidden threats from within.

When it sees something suspicious, it sends a signal to the central command. However, if an attacker finds a way to block that signal using DNS tricks or traffic redirection, the watchtower remains lit and appears normal. Meanwhile, the attacker walks in the shadow.

This is MDI's blind spot. It assumes its signals are always delivered, but a quiet attacker can cut that link and operate freely inside the network (a specific domain controller).

This blog post sheds some light on a real-life scenario that was recently investigated. This scenario was an evasion technique that focuses on traffic manipulation and security telemetry suppression through network redirection. Specifically, it targets the MDI sensor's reliance on local DNS resolution and the unverified assumption that outbound TLS traffic consistently reaches Microsoft's backend. By redirecting sensor traffic to a local endpoint using DNS tampering, an attacker can keep the sensor service running while silently severing telemetry transmission. This renders the sensor blind, without generating alerts or impacting operational continuity.

The technique is “OLD BUT GOLD”. Simple in execution yet highly effective, this has been observed in real world incidents where advanced threat actors and red teams exploited DNS trust to quietly eliminate Defender for Identity visibility while maintaining the appearance of a healthy sensor. This approach requires no external binaries, no code injection, and no service disruption.

Additionally, in the real scenario, the attacker made a few more actions, such as configuring the built in firewall on the Windows server to block certain connections.

Some questions were raised, like why didn’t the EDR see many actions? But this is for a different blog post. In the meantime, we’re operating within the AD DS (Active Directory Domain Services) layer, where MDI is actively monitoring and collecting identity related signals.

During my incident investigations and research for this post, I identified multiple techniques to fuzz MDI URL endpoints and discovered a significant number of targets in the process.

Introduction

MDI’s sensor is not just another event log parser. It operates at the intersection of network telemetry, Windows Security Auditing, and Active Directory state introspection, embedding itself as a privileged observer on domain controllers. Its core function is to capture, parse, and correlate low-level protocol behavior such as Kerberos ticket exchanges, NTLM challenges, LDAP bind requests, and SMB session flows with high-fidelity log signals from the Windows Event Log stack.

At the core of this architecture is Microsoft.Tri.Sensor.exe, which communicates with Microsoft’s cloud through tenant-scoped API endpoints such as sensorapi.atp.azure.com. This communication includes structured telemetry, behavioral signals, heartbeat status, and version metadata. These interactions form the foundation of MDI’s anomaly detection pipeline.

The core sensor engine (Microsoft.Tri.Sensor.exe) operates at the system level and leverages:

- **Npcap-based packet interception**, bound directly to DC interfaces, captures real-time authentication data. Utilizes to capture and analyze network data. The sensor and Npcap work together to provide comprehensive threat detection capabilities.
- **ETW (Event Tracing for Windows)** and **WMI providers** to access local security and directory events. Specifically, it utilizes ETW for real-time event logging and WMI providers to access and interpret system information, enabling comprehensive security monitoring and analysis.
- **Telemetry encryption and compression layers** that forward parsed observables to **tenant-scoped backend endpoints** like sensorapi.atp.azure.com or purplexlabsensorapi.atp.azure.com. The sensor encrypts the telemetry data it collects before sending it to the cloud. This ensures that even if intercepted, the data remains unreadable without the decryption key.

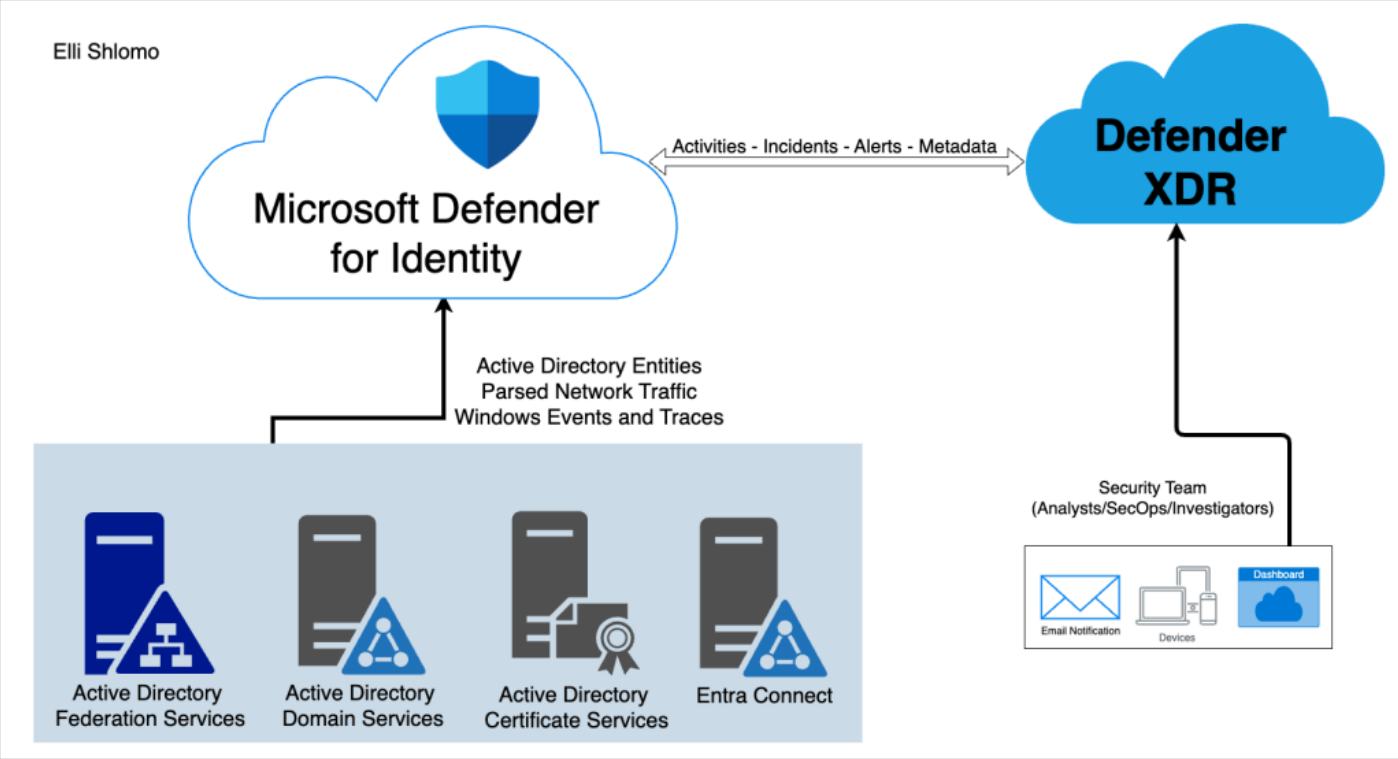
Each sensor is controlled by a cloud command channel and maintains **bidirectional TLS channels** for telemetry, heartbeats, policy updates, and behavioral signal delivery. The telemetry pipeline includes:

- **High-fidelity event sequences** (Kerberos logon chains, ticket reuse patterns, group membership deltas)
- **Sensor status indicators** (load, capture state, service errors)
- **Anomaly triggers** (e.g., DCSync detection, reconnaissance enumeration, lateral movement)

For adversaries and red teamers, the sensor represents a **critical point of visibility**. Disabling it outright is noisy and detectable. However, **preserving operational appearance while degrading its telemetry capabilities** offers a stealthy and highly effective approach to sensor evasion.

Note: If you are not monitoring the MDI traffic, services, and infrastructure, you have a blind spot.

Microsoft Defender for Identity – High-Level Architecture



Sensor Internal

At the heart of Microsoft Defender for Identity (MDI) lies a distributed sensor-based architecture, where each Domain Controller hosts a self-contained runtime that captures, parses, and forwards telemetry to the Microsoft cloud. This runtime is composed of two tightly coupled executables: **Microsoft.Tri.Sensor.exe** and **Microsoft.Tri.Sensor.Updater.exe**.

Note: The sensor discussed here is classified as a legacy component. For modern deployments, Microsoft recommends transitioning to the unified Defender sensor for enhanced coverage and integration.

These components establish **secure outbound communication over HTTPS**, primarily targeting sensorapi.atp.azure.com or tenant-specific subdomains (e.g., purplexlabsensorapi.atp.azure.com). The

API serves as the authoritative control plane for sensor-to-cloud interaction, enabling the ingestion of real-time telemetry, health monitoring, and update coordination.

The Core Telemetry Engine

The Microsoft.Tri.Sensor.exe executable is the operational backbone of the MDI sensor. It runs under the AATPSensor Windows service and performs multiple low-level operations in parallel:

Network Inspection

Packet Inspection: Hooks into the network stack to inspect raw traffic at the transport layer. It parses authentication protocols, such as Kerberos, NTLM, LDAP, and SMB, directly from the domain controller's live traffic, eliminating the need for a network tap or mirror port.

Security Event Monitoring: Actively subscribes to Windows Event Logs, specifically targeting identity-related events such as 4624 (logon), 4768 (TGT request), 4776 (NTLM authentication), and 4662 (directory access), using ETW or standard event API consumption.

Directory Interaction: Queries local Active Directory for metadata on users, groups, trust relationships, and service accounts to build a real-time graph of identity relationships and privilege flows.

Telemetry Aggregation and Dispatch: Collects and packages telemetry data, detection signals, behavioral anomalies, and health metrics. This data is then transmitted to the tenant-specific endpoint, such as sensorapi.atp.azure.com, over encrypted outbound channels.

Health and Heartbeat Signals: Sends periodic status beacons that report version information, operational status, and connectivity metrics to Microsoft's backend.

Note: for more information about the MDI event collection

Telemetry Serialization

- Structures captured events into telemetry packets
- Applies tenant-specific policy filters and suppression rules
- Encrypts telemetry payloads using TLS

Data is sent to the following URL endpoint

https://<tenant>sensorapi.atp.azure.com/<specific api telemtry>

If unreachable, it enters an exponential backoff retry loop.

Attacker Objective

While there are many ways to disrupt, evade, or outright terminate Microsoft Defender for Identity (MDI) sensors, some of them can cause noise service crashes, telemetry gaps, or obvious health alerts in the

portal. But what if you could manipulate the sensor without ever triggering an alert?

Create a **stealthy, persistent disruption** of the MDI sensor's outbound telemetry **without terminating**, **crashing**, or tampering with it `Microsoft.Tri.Sensor.exe` directly. The goal is simple:

- **Let the process live** – The MDI sensor (`Microsoft.Tri.Sensor.exe`) remains up, running, and healthy. No service crash. No recovery trigger. The process thinks it's doing its job.
- **Break the signal flow** – By routing `sensorapi.atp.azure.com` and related endpoints to `127.0.0.1`You've null-routed the telemetry. The sensor transmits, but the packets are lost due to a black hole. Outbound TLS fails silently. No exfiltration. No alerts.
- **Stay below detection thresholds** – Because you're not terminating services, uninstalling software, or dropping connections via firewall rules, you don't trigger: Defender for Identity sensor health alerts or SIEM rules built on service failure events.

Microsoft Defender for Identity sensors **do not validate** whether the data they send is *actually received* by Microsoft's backend. There's no:

- End-to-end ACK or receipt
- Mutual TLS pinning from the agent
- DNS validation guardrails

They trust DNS, like many other security controls, such as EDR. IN the EDR world, there is a known tactic and manipulation.

Traffic Manipulation

Stage: DNS Redirection

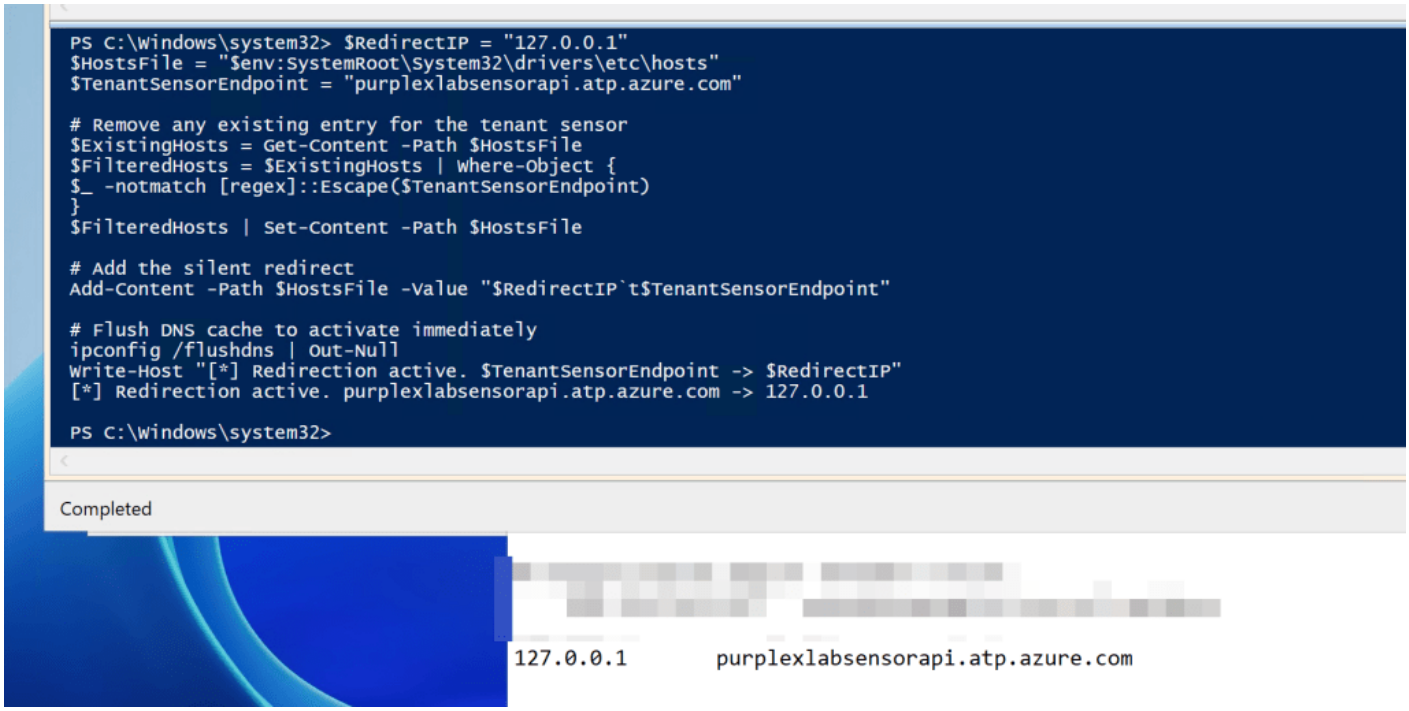
We will redirect telemetry endpoints to 127.0.0.1, null-routing the connection.

```
$RedirectIP = "127.0.0.1"
$HostsFile = "$env:SystemRoot\System32\drivers\etc\hosts"
$TenantSensorEndpoint = "purplexlabsensorapi.atp.azure.com"

# Remove any existing entry for the tenant sensor
$ExistingHosts = Get-Content -Path $HostsFile
$FilteredHosts = $ExistingHosts | Where-Object {
$_ -notmatch [regex]::Escape($TenantSensorEndpoint)
}
$FilteredHosts | Set-Content -Path $HostsFile

# Add the silent redirect
Add-Content -Path $HostsFile -Value "$RedirectIP`t$TenantSensorEndpoint"
```

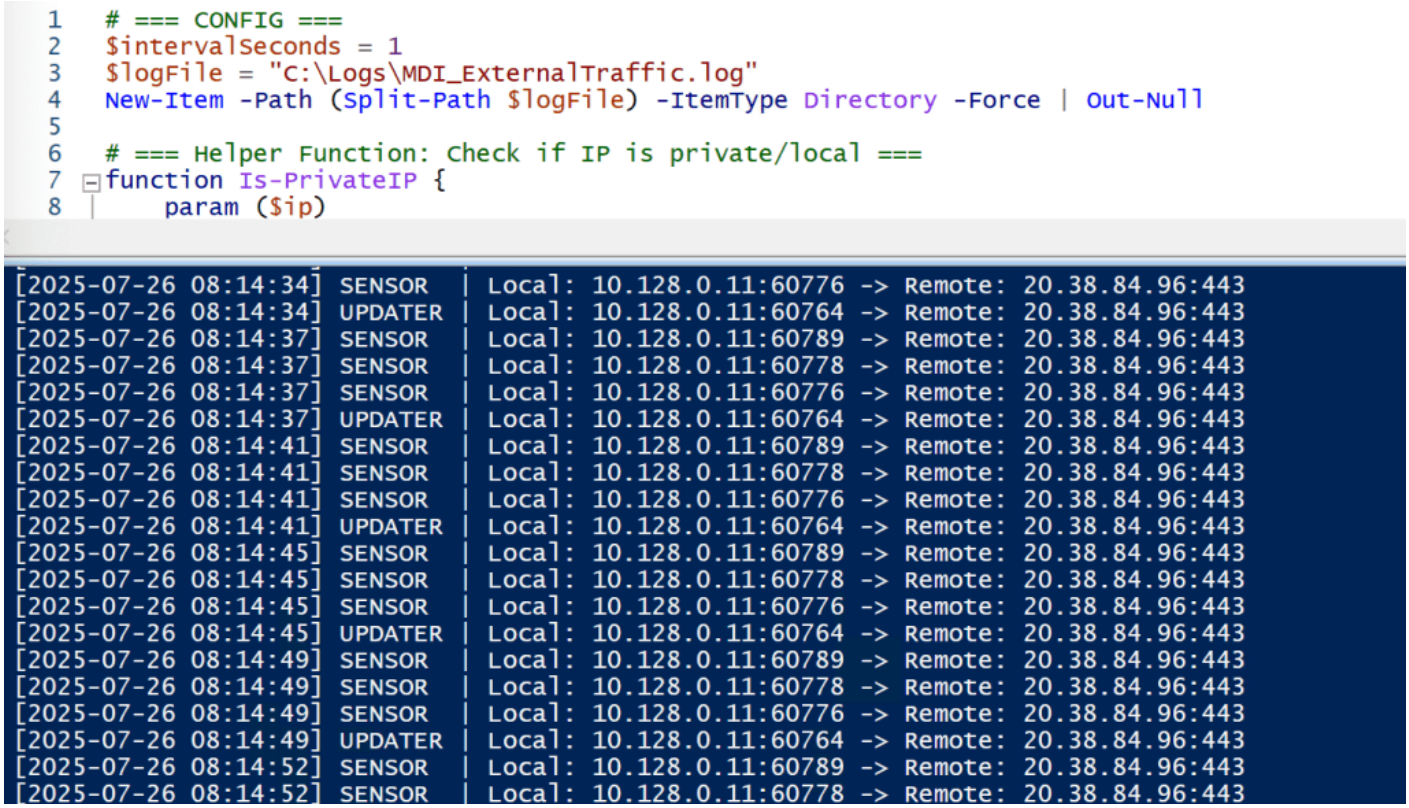
```
# Flush DNS cache to activate immediately
ipconfig /flushdns | Out-Null
Write-Host "[*] Redirection active. $TenantSensorEndpoint -> $RedirectIP"
```



Note: You should configure your workspace URL endpoint.

Stage: External Traffic Monitor for MDI Sensor

Before the changes, and when the routing remained unchanged.



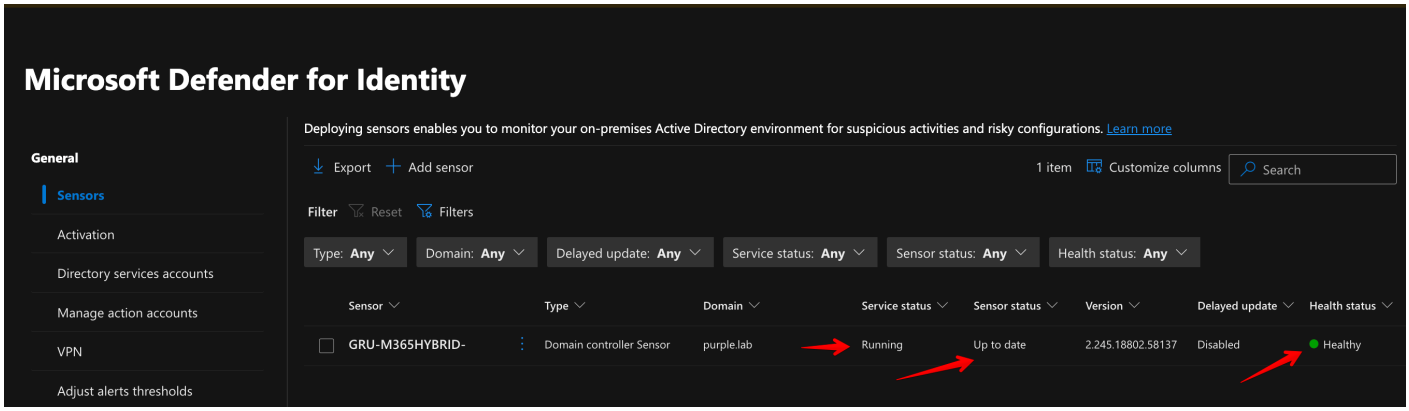
After the changes, and when the routing is for the local 127.0.0.1

```
  6   # === Helper Function: Check if IP is private/local ===
  7   function Is-PrivateIP {
  8       param ($ip)
```

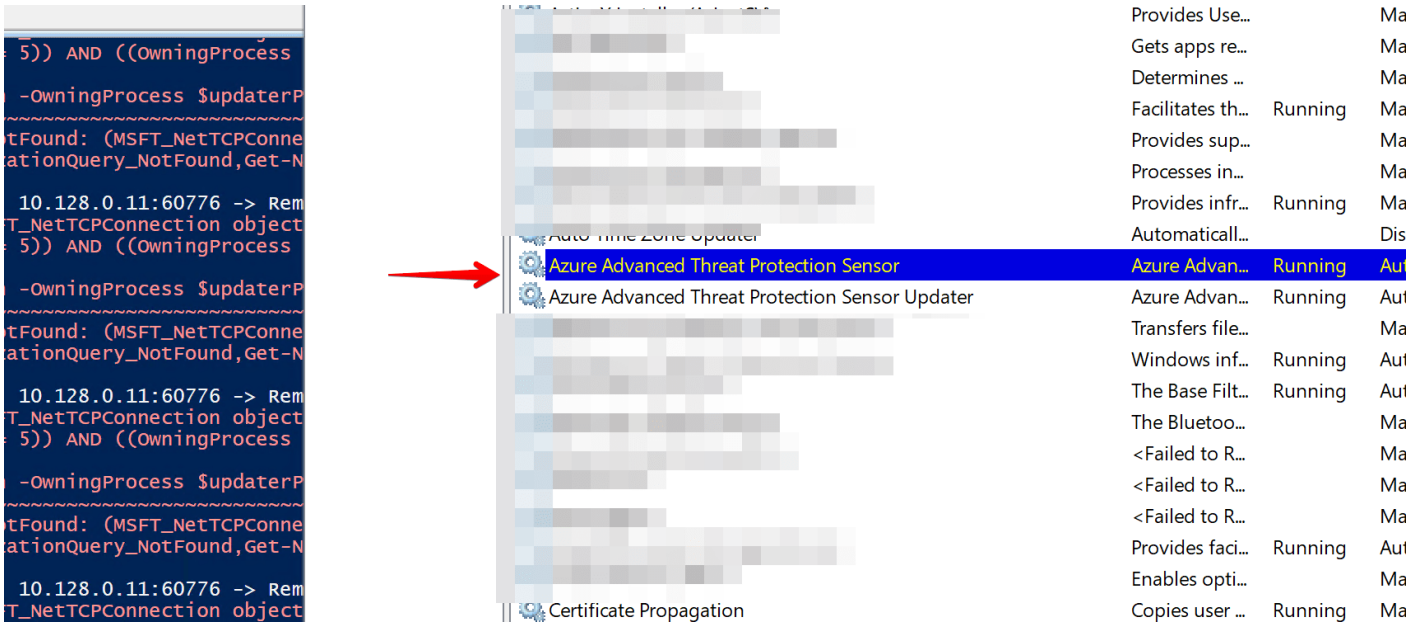
```
[2025-07-26 08:17:38] SENSOR  | Local: 10.128.0.11:60776 -> Remote: 20.38.84.96:443
Get-NetTCPConnection : No matching MSFT_NetTCPConnection objects found by CIM query for instances of the ROOT/StandardC
MSFT_NetTCPConnection  WHERE ((State = 5)) AND ((OwningProcess = 6172)). Verify query parameters and retry.
At line:32 char:25
+ ... aterConns = Get-NetTCPConnection -OwningProcess $updaterProc.Id -Stat ...
+                 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (MSFT_NetTCPConnection:String) [Get-NetTCPConnection], CimJobException
    + FullyQualifiedErrorId : CmdletizationQuery_NotFound,Get-NetTCPConnection

[2025-07-26 08:17:42] SENSOR  | Local: 10.128.0.11:60776 -> Remote: 20.38.84.96:443
Get-NetTCPConnection : No matching MSFT_NetTCPConnection objects found by CIM query for instances of the ROOT/StandardC
MSFT_NetTCPConnection  WHERE ((State = 5)) AND ((OwningProcess = 6172)). Verify query parameters and retry.
At line:32 char:25
+ ... aterConns = Get-NetTCPConnection -OwningProcess $updaterProc.Id -Stat ...
+                 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (MSFT_NetTCPConnection:String) [Get-NetTCPConnection], CimJobException
    + FullyQualifiedErrorId : CmdletizationQuery_NotFound,Get-NetTCPConnection
```

The “Identities” section in the Defender XDR portal continues to display a healthy status, even hours after the disruption was introduced.



The MDI services remain fully operational and show no signs of disruption.



Assumptions vs. Reality

From the attacker's perspective, Defender for Identity's architecture is rich with trust assumptions about DNS resolution, outbound network communication, and the integrity of the local sensor. Each of these assumptions can be exploited silently using native PowerShell, local privileges, and protocol-level deception without stopping services, writing malware, or triggering local defenses.

Below are the critical assumptions and how a capable attacker dismantles each one.

Assumption: If the AATPSensor service is running, the sensor is fully operational.

The reality:

- Keep the service running, but null-route its telemetry.
- Use DNS redirection (hosts file override) to silently forward sensorapi.atp.azure.com to 127.0.0.1.
- The service logs show normal operation, but **no data reaches Microsoft**.

Assumption: DNS resolution is trustworthy and reflects accurate endpoint routing.

The reality:

- Tamper with the local DNS resolver stack.
- Modify C:\Windows\System32\drivers\etc\hosts to override Microsoft endpoints.
- Exploit the sensor's lack of endpoint verification, it does **not validate remote TLS fingerprints or pinned DNS names**.

Assumption: Sensor telemetry sent over TLS is encrypted, reliable, and visible to Microsoft.

The reality:

- Break the outbound path while maintaining a local TCP handshake.
- Redirect to a local HTTPS listener on port 443 that responds with HTTP 200 OK to /heartbeat.
- This **spoofs backend liveness** and maintains the illusion of connectivity.

Assumption: MDI sensors are monitored in real-time by the SOC.

The reality:

- Exploit the **backend heartbeat delay**
- The portal may show "healthy" status even while telemetry is fully blind.
- Complete all lateral movement and privilege escalation **within the blind window**.

Assumption: Defender for Endpoint or Microsoft Security Copilot will detect sensor tampering.

The reality:

- Avoid service termination, memory injection, or binary replacement.
- Use **PowerShell-only native techniques** that blend into administrative baselines.
- No Defender alert, no Event ID 7045, no EDR signature triggered.

Each assumption is a **point of trust**. Each point of trust, when left unmonitored, becomes a **blind spot**. An attacker doesn't need to defeat the sensor. They only need to **turn it against itself,** operational, quiet,

and completely blind.

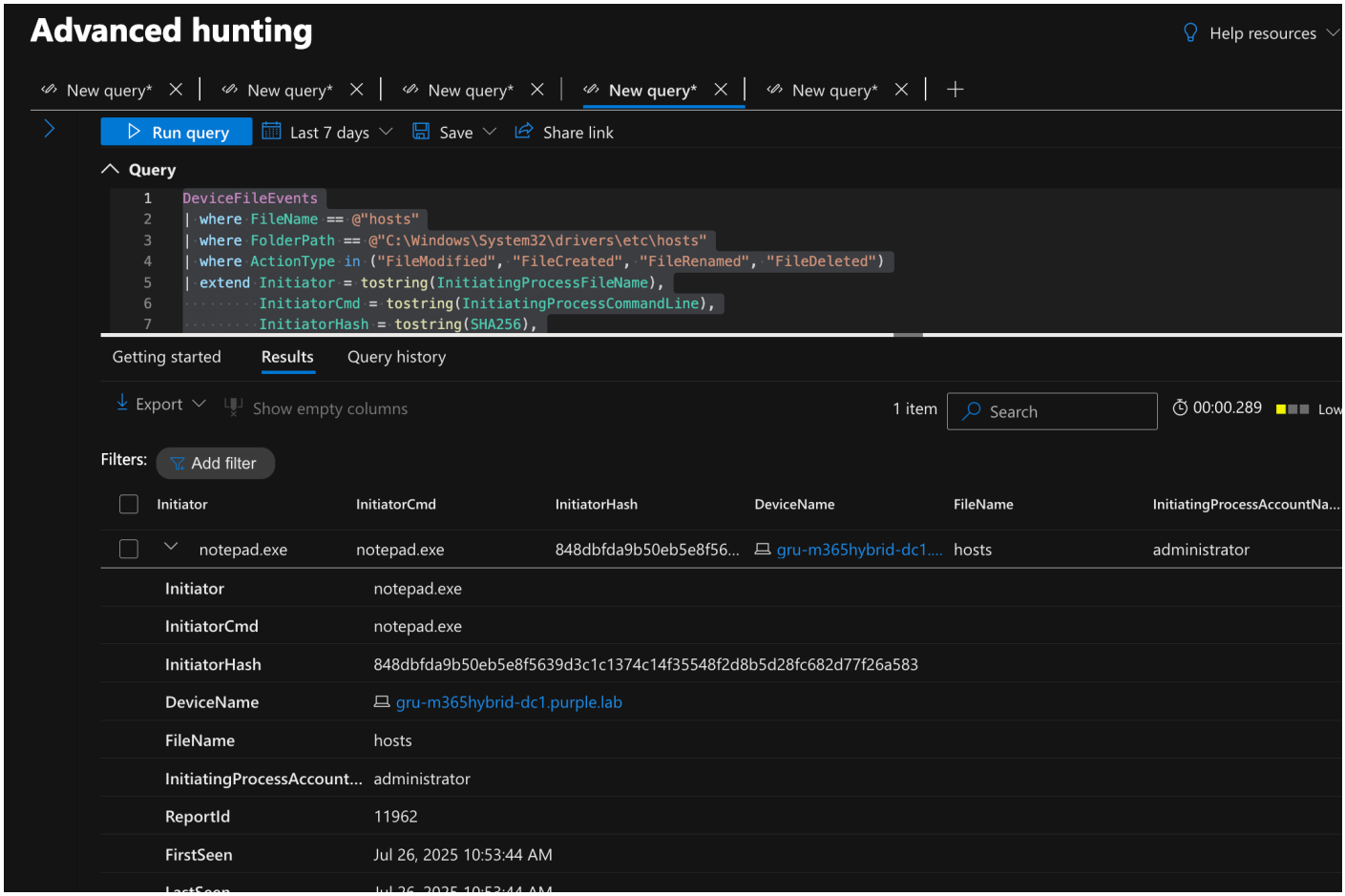
The Blue Team’s Side

Signal Source	Detection Vector
hosts file monitoring	File integrity monitor, Defender for Endpoint
Net logs	Lack of outbound 443 to *.atp.azure.com
MDI Log Inactivity Detection	checks key MDI telemetry tables for recent activity
Sysmon Event ID 1	Unusual PowerShell modifying system files

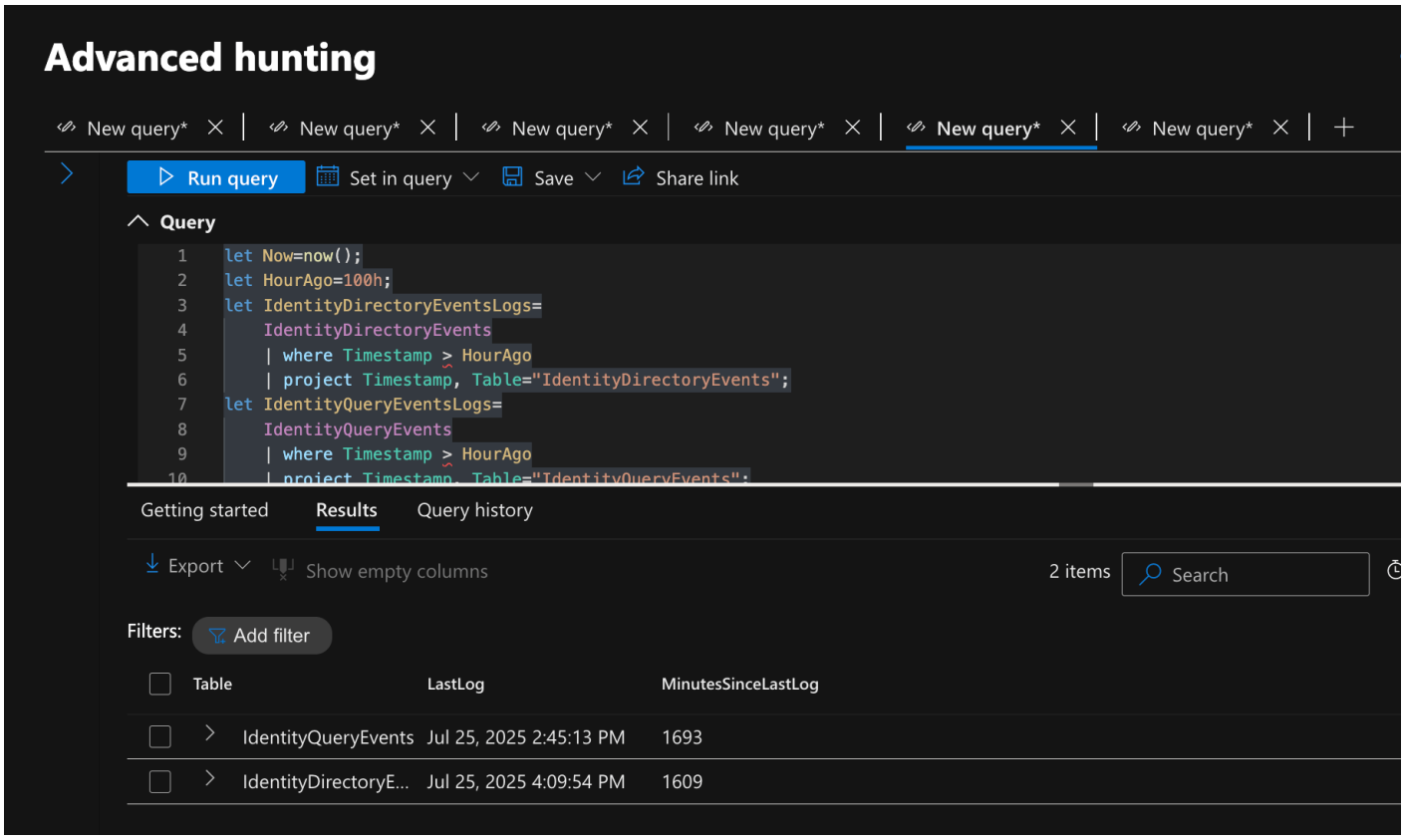
Hosts file monitoring

If you’re using Microsoft Defender for Endpoint, Advanced Hunting provides a powerful way to monitor, investigate, and detect any suspicious modifications to the hosts file in real time.

Hosts File Tampering Detection – Process Attribution and Timeline



MDI Log Inactivity Detection



Conclusion

DNS layer sensor disruption offers a powerful and stealthy method of evading Microsoft Defender for Identity telemetry **without stopping services**, **altering binaries**, or triggering **local tampering alerts**.

From an offensive security standpoint, this method is ideal for:

- Simulated domain takeovers
- Low-and-slow lateral movement scenarios
- Cloud disconnect testing
- Sensor visibility boundary assessment

For defenders, it reinforces the critical need for:

- Proactive heartbeat monitoring
- System file integrity enforcement
- Network-layer validation of telemetry flow

If your sensor is running, but MDI never sees it, you are blind.