Nightfall AI

NIGHTFALL AI

STATE OF AGENTIC DATA SECURITY 2026

88.9M MCP EVENTS OBSERVED

608 DISTINCT MCP PACKAGES IDENTIFIED

NIGHTFALL.AI

CONTENTS

What's inside

01 Executive Summary

The rapid adoption of AI agents is creating a new visibility and governance challenge. Through the MCP servers they connect to, agents can access source code, credentials, production databases, cloud infrastructure, email, calendars, design systems, HR platforms, finance tools, and CRM data. Much of this activity occurs outside the controls designed to monitor human-driven data movement.

This report is based on observed activity, not survey responses. Across 88.9 million MCP events in Nightfall's enterprise customer base over a 3-month period, we identified 608 distinct MCP-associated packages. The largest is operating almost 200 MCPs. Average MCP tool-call volume per organization grew nearly 4× from June to July 2026.

A new class of data mover

For two decades, data loss prevention has rested on one assumption: a person moves data. Someone attaches a file, pastes into a browser, or sends a message, and a control inspects that action as a human takes it. Every mature category inherits the assumption — endpoint DLP watches the user's device, CASB and SSE inspect the user's traffic, email DLP scans what the user sends.

AI agents break it. An agent is a process that holds credentials, chains tool calls, and moves data between systems at machine speed without a human initiating each step. When an agent reads a repository, queries a production database, and writes the result into a ticket, no upload occurred, no paste occurred, and no message was sent. The data moved anyway. The Model Context Protocol makes this possible at enterprise scale — one protocol, hundreds of servers, access to source control, cloud infrastructure, databases, communications, and knowledge bases — so the risk arrives uniformly across the estate rather than app by app.

Why the incumbent categories cannot close the gap

The gap is architectural, not a matter of feature maturity. Legacy DLP is pattern-matched to human gestures: it can classify a file and block an upload, but it has no concept of a tool call, no record of which server initiated an action, and no view of what a tool returned. An agent that never uploads anything never trips it.

Network-delivered controls — secure web gateways, CASB, SSE, and the emerging MCP gateway category — can only act on traffic that crosses a network boundary, and much MCP activity never does. Where traffic does cross, a gateway sees an ordinary API call to an expected domain, not the agent behind it or the payload's meaning. Endpoint and EDR tooling can see a process start, but process telemetry is not data telemetry. Posture and registry tools inventory what was approved rather than what is running — and this report's central finding is that the two lists differ. Identity systems can scope a grant but cannot inspect tool arguments or returned content.

The result is a category-wide blind spot. No control plane in production today covers discovery of local servers, inspection of configuration files, semantic visibility into tool calls, and pre-execution enforcement at once.

A second threat surface: the agent's own trust boundary

Agents routinely process untrusted content — webpages, repository contents, documents, emails, and tool responses — and any of it can carry an instruction. A payload arriving through one tool response can influence the agent to read from another connected system, expose a credential, or take an unauthorized action. The vulnerable component and the sensitive resource need not share a trust domain, and a trusted parent process is not a trust boundary. The observed data shows this is not a future exposure: credential-format strings sat in configuration files in nearly half of organizations, 1 in 5 routed official traffic through an unverified proxy, and 12 or more agent clients were launching servers.

What the data shows

- **Local MCP servers are widespread and largely invisible to network controls.** A browser-automation server was observed in nearly half of organizations, Chrome DevTools in approximately 40%, Docker's MCP plugin in 35%, a debugger server in 30%, and a containerized GitHub server in 25%. Because these exchanges occur locally, the traffic may never cross a gateway or proxy.
- **~70% of organizations were running at least 1 unsupported MCP server.** @modelcontextprotocol/server-postgres, last published in December 2024 and marked "no longer supported," was present in roughly 25% of organizations. Some of these deprecated packages remain connected to production databases, repositories, and collaboration systems.
- **2 in 3 observed MCP packages had no verified publisher.** Security teams often cannot establish who produced a package, whether it is still maintained, or whether the installed software corresponds to the expected vendor.
- **Nearly 50% of organizations exposed credentials in MCP configuration files.** Environment-variable indirection—the documented alternative—appeared in fewer than 1 in 10 credential-bearing configurations. MCP configuration files are becoming a new, largely unmonitored credential store.
- **At least 1 in 5 organizations routed an official MCP connection through an unverified intermediary.** Agents reached Atlassian's official MCP endpoint through mcp-remote, a community proxy maintained by 2 individuals and lacking a verified publisher. The final destination appeared legitimate, while the additional trust layer remained largely invisible.
- **12+ agent clients were observed launching MCP servers.** The ecosystem extends well beyond Claude Code, Cursor, and VS Code to Codex, Zed, ChatGPT, cmux, PyCharm, IntelliJ IDEA, WebStorm, and other development and orchestration tools. Governing 1 client does not govern the broader MCP surface.
- **No single control plane covers the full MCP lifecycle.** Gateways can inspect remote traffic but cannot see local servers or configuration files. Endpoint controls can discover local activity but may not cover unmanaged devices. Runtime hooks can inspect tool calls but must be deployed separately by client. Identity, authorization, registry, and compliance systems each address only part of the problem.

The remainder of this report translates these findings into a threat matrix, a control-plane coverage matrix, 11 control requirements, and an MCP Security Readiness Checklist. The foundational question remains: Which MCP servers are currently running across the organization—and can risky actions be stopped before they execute?

02 The blind spot

Nearly 50% of organizations run a local MCP server the network may never see

A network control plane sees only traffic that crosses a network boundary. MCP activity does not always do that.

Remote MCP servers communicate across the network, where their traffic can be routed through a gateway or proxy. Local MCP servers are different. They run as child processes on the same device as the agent host and commonly communicate over standard input and output. The prompts, tool requests, query results, file contents, and credentials exchanged between the agent and server may never leave the endpoint.

As a result, a gateway that sees every remote MCP connection may still be blind to a large share of actual MCP activity.

Local servers were common across the organizations observed.

TABLE 1. LOCAL MCP SERVERS OBSERVED RUNNING, BY SHARE OF ORGANIZATIONS.

LOCAL MCP SERVER	SHARE OF ORGANIZATIONS
Browser automation (playwright-mcp)	~45%
Chrome DevTools (chrome-devtools-mcp)	~40%
Docker MCP plugin (docker-mcp)	~35%
Debugger (lldb-mcp)	~30%
Containerized GitHub (ghcr.io/github/github-mcp-server)	~25%

The control gap is architectural. A network product cannot inspect traffic that never reaches the network. Any control plane that depends on traffic passing through a proxy is structurally blind to local MCP activity.

Some local servers make outbound calls that a gateway can observe. But what the gateway sees is an ordinary API request from the host—not which MCP server initiated it, which agent invoked the server, what tool was called, or what data was returned.

Why a trusted parent is not a trust boundary

The parent process identifies which application launched an MCP server. It does not establish what the server can access, who published it, whether it is maintained, or how it will behave.

This distinction matters because agents routinely process untrusted content: webpages, repository contents, documents, emails, tool descriptions, and tool responses. An MCP server can introduce instructions into the agent's context, and those instructions may influence later tool calls or data access.

The resulting trust chain is longer than it appears:

```
Untrusted content → agent context → MCP tool call → sensitive system
```

A prompt-injection payload arriving through one tool response can influence the agent to read from another connected system, expose a credential, or perform an unauthorized action. The vulnerable component and the sensitive resource do not need to share the same trust domain.

03 Why Legacy DLP is blind to MCP activity

Agentic data movement does not follow a human-controlled path

Legacy DLP was designed around recognizable human actions: uploading a file, pasting into a webpage, sending an email, copying to removable media, or moving content through a managed SaaS application.

Agentic data movement follows a different path.

An agent can read a local file, query a database, retrieve source code, call an API, transform the result, and send it to another tool without a human manually initiating each step. The movement occurs through tool calls and agent runtimes rather than the upload, paste, and send actions traditional DLP was designed to monitor.

Legacy DLP generally does not understand:

- Which MCP server initiated an action
- What arguments were passed
- Whether the destination was expected
- Whether an instruction originated from an untrusted tool response
- Which tool was called
- What data the tool returned
- Whether retrieved data influenced a later action

Modern data security must cover both human and agent activity, with runtime context about the agent, tool, destination, action, and data involved.

Without that context, a control may see an API request or process execution while missing the agentic workflow that produced it.

04 The MCP surface

MCP is already an enterprise-wide stack, not a single developer tool

MCP adoption spans development, communications, knowledge management, design, observability, cloud infrastructure, databases, finance, and go-to-market systems.

The most widely reached remote MCP endpoints were not limited to AI-native platforms. They included enterprise services such as Atlassian, GitHub, Microsoft, Google, Slack, Notion, Figma, Docker, Linear, GitLab, Asana, and Datadog.

TABLE 2. MOST WIDELY REACHED REMOTE MCP ENDPOINTS, MAY–JULY 2026.

ENDPOINT	SHARE OF ORGANIZATIONS	CONNECTIONS
Anthropic MCP proxy	~75%	526,167
Atlassian	~75%	7,195,970
GitHub Copilot API	~75%	192,851
Microsoft Learn	~75%	17,203
Google Firestore	~70%	1,288,134
Slack	~70%	310,219
Notion	~65%	308,008
Hugging Face	~65%	20,771
Figma	~65%	301,082
Docker	~60%	171,977
Linear	~60%	309,761
GitLab	~55%	2,754,525
Asana	~50%	28,817
Datadog	~50%	85,142

The breadth of these connections matters. MCP is not confined to experimental developer environments. It is becoming an access layer across the systems where organizations store source code, credentials, internal knowledge, operational telemetry, customer data, and business records.

TABLE 3. MCP SERVER CATEGORIES BY SHARE OF ORGANIZATIONS. FUNCTIONAL CATEGORY IS OBSERVABLE FROM THE SERVER ITSELF; ASSIGNMENT IS AN ANALYST PASS OVER A FIXED VENDOR LIST.

CATEGORY	SHARE OF ORGANIZATIONS
Docs & knowledge	~90%
Work management	~85%
Source control & CI	~80%
AI platform	~75%

CATEGORY	SHARE OF ORGANIZATIONS
Communications	~75%
Databases & data stores	~70%
Design	~65%
Cloud & infrastructure	~60%
Finance & GTM	~60%
Observability & ops	~55%

Assignment to a functional category is based on the server itself. Individual implementations may span multiple functions or expose access to more than one system.

04.1 Approximately 70% of organizations run at least 1 unsupported MCP server

Unsupported MCP software is already present in production environments.

@modelcontextprotocol/server-postgres was last published in December 2024 and is explicitly marked “no longer supported.” It was observed in approximately 25% of organizations, including environments where it was connected to production databases.

The package was one of several reference implementations deprecated after the protocol’s launch. Of the 36 npm-packaged servers observed, 6 were formally deprecated. All 6 were reference implementations from the protocol’s original scope.

These packages were intended as examples, but many organizations adopted them as production infrastructure. As the ecosystem moved toward maintained vendor and community implementations, deployed configurations did not always move with it.

Deprecation is not the same as a known compromise. It is a maintenance risk. Unsupported packages stop receiving security fixes, compatibility updates, dependency maintenance, and protocol revisions while often retaining access to high-value systems.

04.2 2 in 3 observed packages have no verified publisher

Package provenance is a separate problem from maintenance status.

Of the 36 npm-packaged MCP servers observed, 25 had no verified publisher. An unverified package may be current and functional, but administrators have less assurance about who produced it or whether it corresponds to the vendor or project it claims to represent.

TABLE 4. OBSERVED MCP SERVER PACKAGES BY MAINTENANCE AND VERIFICATION STATUS.

PACKAGE	SHARE OF ORGS	LAST PUBLISHED	STATUS
@modelcontextprotocol/server-github	~45%	2025-04	Deprecated
@modelcontextprotocol/server-slack	~25%	2025-04	Deprecated
@modelcontextprotocol/server-postgres	~25%	2024-12	Deprecated
@modelcontextprotocol/server-gdrive	~15%	2025-01	Deprecated
@modelcontextprotocol/server-puppeteer	~15%	2025-05	Deprecated
@modelcontextprotocol/server-brave-search	~5%	2024-12	Deprecated
mcp-remote	~65%	2026-02	Current, no verified publisher
@upstash/context7-mcp	~50%	2026-07	Current, verified
@modelcontextprotocol/server-filesystem	~35%	2026-07	Current, verified
@modelcontextprotocol/server-sequential-thinking	~30%	2026-07	Current, verified
@notionhq/notion-mcp-server	~15%	2026-07	Current, no verified publisher
@playwright/mcp	~10%	2026-07	Current, verified
@sentry/mcp-server	~5%	2026-07	Current, verified

An abandoned package and an unverified package present different risks:

> **Abandoned:** The software is no longer actively maintained.

> **Unverified:** The identity of the publisher has not been independently established.

> **Abandoned and unverified:** Administrators may have neither ongoing maintenance nor reliable provenance.

Package names alone are not sufficient evidence of trust.

04.3 1 in 5 organizations route official MCP traffic through an unverified intermediary

Official does not always mean direct

In at least 1 in 5 organizations, agents reached Atlassian's official MCP endpoint through mcp-remote, a community proxy maintained by 2 individuals and published without a verified publisher.

The connection path was:

```
Agent client → mcp-remote → mcp.atlassian.com
```

A network control watching only the final destination sees a connection to an expected Atlassian domain. It may not reveal that an additional package sits in the path and can observe or process the connection.

This does not establish malicious behavior by the intermediary. It demonstrates that trust cannot be determined from the destination domain alone. Organizations also need visibility into the software that establishes, proxies, or transforms the connection.

04.4 12+ agent clients launch MCP servers

The MCP client ecosystem extends well beyond the most commonly governed IDEs.

Servers were launched by at least 12 agent clients beyond the expected concentration in Claude Code, Cursor, and VS Code. Codex appeared in approximately 40% of organizations; Zed, ChatGPT, and cmux in approximately 35%; and several JetBrains clients in approximately 20%–25%.

TABLE 5. AGENT CLIENTS OBSERVED SPAWNING MCP SERVERS THAT NAME-BASED ATTRIBUTION DID NOT MATCH. A FLOOR — AMBIGUOUS PARENTS ARE EXCLUDED RATHER THAN COUNTED.

AGENT CLIENT	SHARE OF ORGANIZATIONS
Codex	~40%
Zed	~35%
ChatGPT	~35%
cmux	~35%
Code (VS Code variant build)	~30%
PyCharm	~25%
IntelliJ IDEA	~25%
claude (CLI binary)	~20%
WebStorm	~20%
Antigravity	~20%
conductor	~20%
Xcode	~20%

Agent orchestrators such as cmux and conductor add another layer. A single client process may coordinate multiple concurrent agent sessions, each with its own MCP servers, tools, and permissions.

Controlling 1 client does not control the broader MCP surface. Discovery must account for the full client tail, including IDEs, command-line agents, desktop applications, orchestration layers, and emerging agent runtimes.

Average MCP tool-call volume per organization grew nearly 4× from June to July 2026. The increase was driven by more activity inside existing organizations, not simply by an increase in the number of organizations observed.

05 Credentials in the blind spot

Nearly 50% of organizations have credential-format strings in MCP configuration files

Secret scanning has traditionally focused on source repositories. MCP configuration files create a different exposure path.

Agent configuration files are the first place many users put the credentials required to connect an MCP server to a database, source-code platform, collaboration service, design system, or cloud environment. These files may live in user-scoped or machine-scoped directories that are not committed to source control and are therefore not examined by repository scanners.

Across 1.83 million MCP configuration observations, nearly 50% of organizations had at least 1 file containing a string matching a known vendor credential format.

TABLE 6. CREDENTIAL FORMATS OBSERVED IN MCP CONFIGURATION FILES, BY SHARE OF ORGANIZATIONS.

CREDENTIAL FORMAT	SHARE OF ORGANIZATIONS
GitHub personal access token	~40%
Notion integration token	~35%
PostgreSQL connection string	~30%
Figma access token	~25%
Slack bot or user token	~15%
Anthropic API key	~15%
AWS access key ID	~5%
GitLab personal access token	~5%

A format match does not establish that every value was active at the time of observation. It does show that credential-like material is frequently stored in local MCP configuration files outside the repositories and secret-management workflows where security teams typically look for it.

The observed credential formats also map directly to the services agents are configured to access, including GitHub, Notion, PostgreSQL, Figma, Slack, Anthropic, AWS, and GitLab.

Environment-variable indirection—the documented alternative to embedding a value directly—appeared in fewer than 1 in 10 credential-bearing observations. Approximately 2 in 3 organizations used environment-variable indirection somewhere, but within individual configurations its use was inconsistent.

The safe pattern exists. It is not yet the default.

05.1 Why the safe pattern is difficult to maintain

Credentials in configuration files are not always the result of obvious carelessness. In several cases, the documented setup path itself encourages unsafe handling.

An official GitHub example places a personal access token directly in a configuration header. A PostgreSQL example places a database password in a command-line connection string. These patterns reduce setup friction, but they also create plaintext secrets in configuration files, process arguments, or shell history.

Tooling can also undo an otherwise safe configuration. A reported issue from January 2026 described a command that replaced ${...} environment-variable references with their resolved values and wrote those values back into the shared configuration file.

A configuration helper can therefore become an exposure mechanism. It can turn an environment-variable reference into a stored secret, or turn a local configuration file into a repository artifact that another process later commits.

The control must cover the full lifecycle:

```
Credential creation → configuration → expansion → execution → storage → commit
```

Scanning source repositories addresses only the final stage.

06 Where control lives

No single control plane covers the MCP surface

MCP security is increasingly being designed around 1 control point: the gateway. A gateway is valuable, but it only acts on traffic that crosses a network boundary. Local MCP servers never do.

Other controls cover different portions of the lifecycle:

TABLE 7. CONTROL-PLANE COVERAGE MATRIX.

CONTROL PLANE	DISCOVER LOCAL SERVERS	DISCOVER LOCAL SERVERS	INSPECT CONFIG FILES	ALLOW/DENY REMOTE	TOOL-CALL SEMANTICS	UNMANAGED DEVICES	REAL-TIME BLOCK	AUDIT
Endpoint agent	Yes	Yes	Yes	Partial	Partial	No	No	Yes
Agent runtime- hooks	Partial	Partial	No	Partial	Yes	No	Yes — per call	Yes
Agent runtime -MCP gateway	No	Partial	No	Yes	Remote only	Yes	Remote only	Remote only
Management & policy (MDM, managed config)	No	No	Partial	Yes	No	No	At load only	No
Identity & authorization	No	No	No	Partial	No	Yes	Revocation only	Partial
Compliance / audit API	No	No	No	No	No	—	No	Yes
Registry & provenance	No	No	No	Partial	No	—	No	Partial

The matrix reflects what each control plane can structurally observe or enforce, not the complete feature set of every product available. Endpoint controls and runtime hooks are shown separately because they operate at different instrumentation points and are often deployed independently.

Only the endpoint can reliably discover local MCP servers and inspect local configuration files. A gateway gives broad visibility into remote connections and unmanaged devices but cannot reconstruct local tool calls that never reach the network. Runtime hooks can inspect tool intent and block a call before execution, though their coverage depends on client support. Identity systems authorize access but cannot inspect tool arguments or returned content, and registry controls can restrict approved software without observing what a running tool does with data.

> **Each control plane is necessary. None is sufficient on its own.**

Effective governance requires coordinated coverage across the endpoint, agent runtime, network, identity layer, management plane, and software supply chain.

07 What full coverage requires

11 requirements for governing MCP activity

The following requirements are grouped by the outcome they provide rather than by product or deployment sequence. Requirements 1–3 establish what exists. Requirements 4–7 provide runtime visibility and intervention. Requirements 8–11 establish coverage, authorization, and centralized governance.

TABLE 8. THE MCP CONTROL REQUIREMENTS. GROUPED BY WHAT THEY ACHIEVE, NOT BY SEQUENCE. REQUIREMENTS 1–3 ARE PREREQUISITES FOR THE REST, AND R4 VERSUS R5 IS THE LINE BETWEEN OBSERVING A TOOL CALL AND BEING ABLE TO STOP IT.

#	REQUIREMENT	PLANE THAT SATISFIES IT	WHY NOTHING ELSE DOES
R1	**Discover local servers**	Endpoint agent	Local servers generate no network traffic
R2	**Discover remote servers**	Endpoint agent; MCP gateway partially	A gateway sees the destination, but not which client reached it or what is brokering the connection
R3	**Inspect configuration files**	Endpoint agent	A config file is a file; no network control sees one
R4	**See which tool ran and what it returned**	Agent runtime hooks; MCP gateway partial	Semantics exist inside the agent, not on the wire
R5	**Stop a single tool call before it executes**	Agent runtime hooks only	Interception must happen in the call path; telemetry reports after the fact
R6	**See sensitive data in the tool call and response**	Agent runtime hooks locally; gateway for remote	The payload — file contents, query results, credentials — passes over stdio on one machine
R7	**Detect an instruction where data was expected**	Agent runtime hooks, MCP gateway for routed remote-server traffic	The trust-boundary collapse in Table 8A is visible only where a response is parsed semantically
R8	**Cover devices with no agent installed**	MCP gateway	An endpoint agent cannot see a host it is not on
R9	**Cover agents running in CI or hosted in the cloud**	MCP gateway or identity and authorization	No endpoint is involved, so neither endpoint telemetry nor local hooks apply
R10	**Reduce blast radius before anything is detected**	Identity and authorization	Scope and token lifetime are set at grant time
R11	**Enforce an approved-server list centrally**	Management and policy plane	Enforcement has to originate somewhere administrators control

Tier access by instrumentability

The relevant distinction is not whether a client is approved by name. It is whether its activity can be observed and controlled.

A client without runtime hooks may still support:

- R1–R3 through endpoint discovery and configuration inspection
- R4 through gateway coverage when traffic crosses the network

What it cannot provide is R5: stopping a tool call before execution.

Clients without pre-execution controls should therefore operate under tighter restrictions. They should not receive unrestricted shell access, credential-bearing servers, production data, or broad access to sensitive systems.

Once the same client moves onto an unmanaged device, even endpoint visibility may disappear. At that point, the organization may retain authorization records or network telemetry without knowing which local servers are running or what the agent is doing with retrieved data.

Operating practice

01 Discover. Inventory every agent and every MCP server actually running — not the assumed list, and not only the servers that reach the network.

02 Vet and pin. Enforce an allow-list centrally. Pin and verify versions, and check maintenance status: a working server is not a supported one.

03 Enforce least privilege. Scope tool access and OAuth grants to the task. Get secrets out of config files.

04 Mediate. Prefer automatic enforcement; reserve human approval for what policy cannot decide.

05 Monitor and enforce. Add agent-layer logging alongside existing DLP, including on the local transport, and inspect tool responses for both sensitive content and injected instructions.

08 MCP Security Readiness Checklist

If you cannot answer yes to most of these today, that is the starting point, not a failure.

DISCOVER

- [ ] How many MCP servers are running across your fleet right now?
- [ ] Which of them are local (stdio) versus remote (HTTP, SSE, or streamable HTTP)?
- [ ] Which have been fingerprint-matched only, versus registry-matched with a verified upstream identity?
- [ ] How many are on your approved list? How many are not?

ESTABLISH TRUST

- [ ] Do you know, for each server, whether it is vendor-official, a reference implementation, or community-maintained—and, for the official ones, whether anything is proxying the connection?
- [ ] When a proxy such as mcp-remote is configured, are you inspecting what it proxies?
- [ ] Is anything on that list deprecated by its own maintainers—and would you know, given that the warning may go to a channel nobody reads?
- [ ] Are credentials sitting in plaintext in MCP configuration files on developer machines—and would you know if your tooling had rewritten a safe reference into a literal value?
- [ ] What is the identity behind each of those credentials, and what does it authorize?

GOVERN AND ENFORCE

- [ ] Have you deployed a centrally managed allowlist of approved servers?
- [ ] Do you know which of your agent clients support hooks, and is access tiered accordingly?
- [ ] Can any agent execute shell commands without a human confirming the action?
- [ ] If an agent session is compromised, what could an untrusted server read—including credentials it could reach and the credentials in its own configuration file?
- [ ] Can you block a single tool call before it executes, without killing the session?
- [ ] Can you inspect the response of a tool call for sensitive data or injected instructions?
- [ ] Do you have an enforcement point that can block a risky agent action in real time, not just log it?

INVESTIGATE

- [ ] Does your DLP program see agent-initiated actions the way it sees human ones?
- [ ] Do you have one investigation surface that correlates agent activity with endpoint and SaaS events?

09 How Nightfall closes these gaps

Each finding in this report maps to a control Nightfall already operates at the endpoint and inside the agent runtime.

01

Local servers no network control can see

~45% OF ORGS RAN A LOCAL BROWSER-AUTOMATION SERVER

Nightfall discovers MCP servers on the endpoint itself — including stdio servers that never cross a network boundary — and attributes each one to the agent client that launched it.

02

Unsupported software still connected to production

~70% OF ORGS RAN AT LEAST 1 UNSUPPORTED SERVER

Every discovered server is resolved to a package, version, publish date, and maintenance status, so deprecated reference implementations surface as an inventory fact rather than a rumor.

03

Packages with no verified publisher

25 OF 36 OBSERVED NPM PACKAGES UNVERIFIED

Provenance is reported alongside the running inventory, and an approved-server list is enforced centrally instead of relying on package names as evidence of trust.

04

Credentials sitting in configuration files

NEARLY 50% OF ORGS HAD CREDENTIAL-FORMAT STRINGS IN MCP CONFIG

Nightfall inspects MCP configuration files in place and applies the same detectors used across SaaS and endpoints, catching secrets before they reach a repository — and flagging safe references that tooling has rewritten into literal values.

05

Official endpoints reached through unverified proxies

AT LEAST 1 IN 5 ORGS ROUTED TRAFFIC THROUGH MCP-REMOTE

Because discovery happens where the process runs, the intermediary is visible in the path — not just the legitimate destination domain a gateway would record.

06

A client tail far wider than the governed IDEs

12+ AGENT CLIENTS OBSERVED LAUNCHING SERVERS

Coverage follows the endpoint rather than a single vendor integration, and clients without pre-execution hooks can be tiered down to restricted access automatically.

07

Tool calls with no semantic visibility

88.9M MCP EVENTS, GROWING NEARLY 4× MONTH OVER MONTH

Runtime instrumentation records which agent invoked which tool, with what arguments, against which server, and what came back — the context legacy DLP has no field for.

08

Sensitive data and injected instructions in tool responses

UNTRUSTED CONTENT → AGENT CONTEXT → TOOL CALL → SENSITIVE SYSTEM

Detection is applied to tool arguments and responses on the local transport, inspecting for both sensitive content and instructions arriving where data was expected.

09

Logging where enforcement was needed

R5: STOP A SINGLE TOOL CALL BEFORE IT EXECUTES

Nightfall can block an individual tool call pre-execution and leave the session running, rather than choosing between an after-the-fact alert and killing the agent.

10

Agent activity siloed from the DLP program

HUMAN AND AGENT PATHS MONITORED BY DIFFERENT TOOLS

Agent, endpoint, and SaaS events land on one investigation surface, so agentic data movement is governed by the same policies and reviewed in the same queue as human activity.

Gateways, identity systems, and registries remain necessary. Nightfall covers the layers none of them can reach: the local transport, the configuration file, and the tool call itself.

To see which MCP servers are running across your fleet today, contact Nightfall AI at support@nightfall.ai.

10 Method

Figures derive from a single cross-organization event table populated from endpoint-agent telemetry, 1 May – 30 July 2026. Event counts are precise; organization-level results are shares rounded to the nearest five percent, with exact organization counts withheld as commercially sensitive.

The full methodology is available upon request. Please contact Nightfall AI at support@nightfall.ai.

WHERE NIGHTFALL IS DIFFERENT

Coverage across the whole MCP lifecycle, not one control point

Nightfall instruments the endpoint and the agent runtime together, then correlates that activity with the SaaS and network events security teams already collect. The comparison below is drawn from the coverage matrix in section 06.

REQUIREMENT	LEGACY DLP, CASB & MCP GATEWAYS	NIGHTFALL
Local MCP servers	Invisible — no network traffic to inspect	Discovered on the endpoint, including stdio servers that never touch the network
Configuration files	Out of scope; repository scanners never see user-scoped files	Inspected in place, with credential-format detection before a secret reaches a repo
Tool-call semantics	An API request to an expected domain	Which agent, which server, which tool, which arguments, and what was returned
Sensitive data in responses	Classifies files at rest and uploads in flight	Detection applied to tool arguments and tool responses on the local transport
Prompt injection in tool output	No concept of an instruction arriving as data	Tool responses inspected for injected instructions, not just sensitive content
Stopping a risky action	Log after the fact, or kill the whole session	A single tool call blocked before it executes, session intact
Investigation	Agent activity siloed from endpoint and SaaS events	One surface correlating agent, endpoint, and SaaS activity for human and agent alike

Gateways, identity systems, and registries all remain necessary. Nightfall covers the layers none of them can reach — the local transport, the configuration file, and the tool call itself — and unifies agent and human data protection in one program.