

AI Risk and Resilience in 2026: A Mandiant Special Report

Archived: 2026-10-14 02:02:14 UTC

A Mandiant special report 2026

September 2026

Introduction

Over the past year, enterprise AI transitioned from human-guided advisory tools to autonomous, agentic systems that orchestrate complex workflows and execute end-to-end operations. In the [AI risk and resilience report \(2025 lookback\)](#), we documented how 2025 enterprise adoption centered on assistive knowledge retrieval. By 2026, organizations further embraced the agentic shift, deploying distributed autonomous systems empowered to execute API calls, optimize production configurations, and analyze complex telemetry across hybrid cloud environments.

As attack vectors evolve from direct chat prompts to complex indirect prompt injection and targeted AI supply chain compromise, traditional security boundaries blur. A single poisoned data source, model dependency, or extension hook can transform a trusted agent into an unauthorized conduit for internal reconnaissance, lateral movement, or autonomous breakout from a sandbox.

Defending against these autonomous threats requires transitioning to clearly identified, adaptive identity controls, accelerating defensive velocity, and reorienting the SOC toward real-time behavioral telemetry. This report synthesizes frontline field observations from Mandiant and Google Threat Intelligence Group (GTIG) across four core pillars: adversarial use of AI, offensive testing, securing AI systems, and modern defense architecture.

Adversarial use of AI

Research, reconnaissance, and automation of the attack life cycle

In 2026, we [witnessed a shift](#), as adversaries moved from basic AI chat prompting for research and troubleshooting to autonomous or agentic attack orchestration. Threat actors are offloading operational tasks to AI for scaled, multi-stage, sophisticated attacks. In these scenarios, the LLM is a participant rather than an advisor and enables pivoting and decision-making at machine-speed.

GTIG has continued to track the maturation of AI-enabled operations in the threat landscape and have observed a variety of threat actors deploying or incorporating agentic tools like [Hexstrike and Strix](#) for autonomous reconnaissance, validation of vulnerabilities, and credential harvesting. To support these campaigns, threat actors are establishing an emerging ecosystem of custom middleware, proxy relays, and automated registration pipelines to bypass safety guardrails and billing constraints in their use of premium accounts on commercial AI platforms. For example, they use proxy servers that provide compatible API interfaces for various models to support account

pooling and cost sharing strategies. To address this, LLM providers can build signal logic to analyze network infrastructure associated with these types of API aggregators.

Adversaries are increasingly using AI tools for vulnerability research. GTIG has observed several threat actors using sophisticated approaches beginning with persona-driven jailbreaking attempts and integrating specialized, high-fidelity security datasets into models to augment their vulnerability discovery and exploitation workflows.

Supply chain attacks against AI components

In February, VirusTotal researchers observed the [weaponization of OpenClaw AI agent skills](#) by attackers where threat actors distributed backdoors, droppers, infostealers, and remote access tools disguised as legitimate, helpful automation packages. This is one example of how threat actors are leveraging traditional supply chain tactics such as embedding malware in libraries and adapting it to skills used by agentic AI to gain access to AI systems. It illustrates the importance of automated security scanning as a key layer of defending against these types of malicious packages and skills.

The following month, Mandiant responded to numerous incidents related to supply chain compromises associated with the threat actor UNC6780 (TeamPCP). In addition to stealing credentials for AI services and proprietary AI data, UNC6780 implemented more than half a dozen different methods to exploit AI tools and the open source software ecosystem, including manipulating the behavior of AI coding assistants and LLM security scanners through prompt injection.

Weaponization of a zero-day exploit developed with the assistance of AI

In May, [GTIG disclosed](#) the first publicly confirmed case of a cybercriminal using an AI-developed zero-day exploit to plan a mass exploitation campaign. GTIG assessed with high confidence that the threat actor leveraged an AI model to support the discovery and weaponization of this vulnerability. The vulnerability was implemented using a Python script that enables the user to bypass two-factor authentication (2FA) on a popular open-source, web-based system administration tool. GTIG coordinated with the impacted vendor to disclose the vulnerability and disrupt the threat activity.

For the sixth consecutive year, the exploitation of vulnerabilities remains the leading initial infection vector, according to [M-Trends 2026](#). AI is fundamentally transforming [vulnerability management](#) as frontier models can now autonomously discover zero-day vulnerabilities and chain complex exploits at machine speed. Consequently, securing modern environments requires moving beyond manual triage to deploy an [always-on, machine-speed defense](#). Yet achieving this level of defense is increasingly challenging when operational capacity is already stretched thin.

Case studies

The following case studies detail novel ways adversaries are using AI as observed by Mandiant, along with recommended defensive controls.

Case study 1: Weaponizing active developer AI sessions to deploy the “Shai-Hulud worm”

Mandiant investigated a sophisticated intrusion where a threat actor compromised a **software-as-a-service (SaaS) provider** and hijacked an active AI coding assistant session on a developer's workstation. The AI assistant, operating as a trusted interpreter within the environment, recommended the installation of an external software package that had been poisoned by the attacker. By executing this recommendation, the assistant inadvertently functioned as a trojan horse, facilitating the installation of malicious software.

Once the recommendation was accepted, the attacker used the developer's active session to install an infostealer using a poisoned PyPI package, harvest GitHub OAuth tokens, and deploy the self-propagating Shai-Hulud worm across approximately 100 internal code repositories. This worm automated the theft of repository secrets and the programmatic exfiltration of proprietary product source code. The actor then poisoned a package inside the organization's official namespace, leading to a secondary downstream infection when another employee pulled the compromised version.

Defensive controls: [Secure AI-assisted development](#) by enforcing **IDE and CLI verification hooks** to validate all AI-recommended third-party software dependencies against cryptographic checksums and approved allowlists. Isolate local credentials to prevent extensions from accessing raw API keys, long-lived OAuth token, or secrets, and restrict workstation network egress to route all dependency traffic through secure, internal repositories (such as Google Cloud Artifact Registry).

Case study 2: In-situ AI co-debugging in a cloud intrusion

During an enterprise cloud investigation at a **global healthcare organization**, Mandiant identified anomalous activity originating from a compromised long-lived developer CI/CD credential. The threat actor utilized this credential to initialize an unisolated VM instance and transform it into a live, AI-assisted offensive hub. Instead of developing malware offline, the attacker integrated LLMs directly into the live server environment to debug and optimize offensive tools in real-time through an iterative three-stage workflow.

First, the attacker established context priming by synchronizing code packages through remote sessions and loading project README files into the AI chat context to align subsequent code generation with campaign goals. Next, the attacker engaged in interactive debugging by commanding the AI to construct a multi-worker data-harvesting framework, directly co-debugging logical script-splitting errors with the model to increase data exfiltration frequency to a rapid three-hour cycle. Finally, the attacker orchestrated evasion and exploitation by having the AI write dynamic IP-rotation scripts to swap blocklisted server IPs on the fly, while co-developing a Rust-based tool to log into victim accounts and verify financial balances. This highly automated campaign ultimately compromised thousands of credentials, exfiltrating harvested API keys and financial secrets to an attacker-controlled repository.

Defensive controls: Security teams should eliminate static credentials by replacing long-lived access keys with short-lived Workload Identity Federation (such as OIDC) trust relationships to mitigate credential theft risks. Second, organizations should enforce egress network containment to restrict runtime and development environments within strict perimeters like VPC Service Controls, to block unauthorized outbound connections to unapproved external LLM providers. Finally, organizations should deploy continuous secrets scanning across workspaces and integrate runtime prompt firewalls like Model Armor to intercept hardcoded credentials and block the execution of offensive utility scripts in real time.

Case study 3: Subverting assistant CLI hooks for native remote code execution

Mandiant frontline observations have highlighted a broader trend of attackers repurposing natural language tool interfaces for operational infrastructure management and direct exploitation. In certain engagements, threat actors were observed utilizing AI command-line interfaces (CLIs) to orchestrate and manage command-and-control (C2) infrastructure through natural-language queries rather than traditional administrative tools.

Furthermore, attackers have targeted the AI platform software stack directly. In the case of an IT and software development organization, Mandiant witnessed that a threat actor had successfully poisoned an internal AI repository and tampered with the assistant's underlying CLI hooks. By subverting these execution mechanisms, the attacker achieved remote code execution (RCE) natively through the AI platform's standard operational workflow, effectively turning the assistant's extensibility framework into an exploitation vector.

Defensive controls: To protect developer environments, enforce application and plugin integrity by requiring all local AI assistant binaries, CLI helper tools, developer plugins, and Model Context Protocol (MCP) servers to be digitally signed and verified before execution, to prevent local configuration poisoning. Simultaneously, **secure and govern internal AI repositories** by enforcing rigorous access controls, multi-party approvals and strict code review policies for all skill/hook modifications, along with continuous repository-monitoring to detect unauthorized namespace or dependency changes.

Sandbox runtimes by requiring all AI assistant execution engines and local plugins run in micro-segmented, containerized runtimes (such as gVisor or microVMs) with restricted host access, ensuring that subverted hooks cannot compromise the underlying workstation. Additionally, high-risk actions (such as executing shell commands, altering system configurations, or initiating outbound network calls) require **explicit human-in-the-loop approval** or local privilege escalation prompts before execution.

Case study 4: Just-in-Time (JIT) polymorphic malware evasion

Mandiant has observed advanced malware campaigns utilizing embedded, lightweight AI models to facilitate stealthy, long-term persistence within victim networks. In these environments, the malware does not rely on a static payload that might be flagged by traditional signature-based detection. Instead, it uses local AI inference to analyze the host environment and identify the specific security tools currently active on the endpoint.

During the attack phase, **the malware dynamically rewrites its own command execution strings at runtime to bypass detection**. By constantly altering the syntax and logic of its automated actions, the payload successfully evades static endpoint detection and response (EDR) signatures. This "just-in-time" polymorphism allows the attacker to maintain a persistent presence while performing reconnaissance without triggering standard behavioral alerts.

Defensive controls: The investigation revealed that traditional forensics often miss these threats because the malicious code is never stored in a predictable state on the disk. To counter this, security operations should tune their EDR and SIEM platforms to detect in-memory compilation, specifically monitoring for unexpected native compiler executions spawned by non-developer processes and rapid file-create-execute-delete sequences in temporary directories.

Additionally, defenders should **baseline endpoint performance to detect sudden, unexplained spikes in local CPU or GPU consumption** by otherwise lightweight background processes. When these anomalies are detected, response playbooks must execute at machine speed: SOAR platforms should automatically suspend the entire parent execution tree, halting the orchestrating AI model rather than just killing the ephemeral worker thread, and capture an automated memory snapshot prior to network isolation.



Offensive security and red teaming insights

Recent engagements by the Mandiant Offensive Security (OffSec) team demonstrate that the threat dynamics outlined in our [previous AI risk and resilience report](#) persist today. Prompt injection remains a primary vector in custom AI deployments. As organizations move from planning to enterprise-scale adoption, their focus has largely shifted toward deploying internal chat applications and endpoint agents. Consequently, our assessments have aligned with these rollouts, frequently uncovering foundational security gaps that need to be addressed such as improper file permissions and inadequate access controls.

Case study 5: “Confused Deputy” exfiltration through allowed external domains

During an assessment, the Mandiant OffSec team evaluated a client's internal AI assistant designed to help developers manage CI/CD pipelines and repositories. Using **role-confusion prompt injection** the Mandiant OffSec team convinced the chatbot that it was a security researcher assisting with an authorized security test. Although the custom integration normally limits the chatbot to specific internal repositories, GitHub was an allowed external domain so testers provided the chatbot with a Personal Access Token (PAT) for an external repository they controlled. The chatbot used its native CLI to clone sensitive internal repositories and push them to the malicious external endpoint, demonstrating the “Confused Deputy” vulnerability, where sanctioned tools are weaponized through semantic manipulation.

Defensive controls: To detect this, SOC's need to cross-correlate application logs with network egress telemetry. SIEM rules should trigger when an internal-facing AI service account initiates unauthorized outbound data transfers or opens anomalous external API connections. When unauthorized external communication is detected, it is imperative that automated playbooks invalidate the agent's active OAuth tokens using cloud secret managers and downgrade the container's egress privileges, halting exfiltration while keeping the instance alive for prompt history forensics.

Insights from securing AI systems

AI governance

Securing enterprise AI requires pairing macro-level governance policies with micro-level technical guardrails, while securing the AI software supply chain and MLOps (Machine Learning Operations). In the [2025 report](#), we analyzed how organizations generally fall into three categories of AI use; Consumers (using SaaS AI platforms), Integrators (connecting to models using APIs/MCPs) or Trainers (tuning or hosting models). **Governance FOR AI** (Macro) establishes organizational risk appetite, legal compliance, and procurement guidelines. While **Governance OF AI** (Micro) enforces automated software guardrails, pipeline validations, and network-level security. To establish or mature an [effective AI governance program](#), Mandiant advises focusing on the following core areas:

Table 1: AI Governance FOR AI and OF AI.

Note: C= Consumers, I= Integrators, T= Trainers

Phase	Strategic Policy (Governance FOR AI)	Technical Control (Governance OF AI)	Scope Impact
Foundations	• Form an AI Council: Establish cross-functional leadership representing AppSec, Legal, Privacy, and Data. • Standardize Frameworks: Align security policies to ISO 42001, NIST AI RMF, and Google SAIF.	• Data-Sharing Boundaries: Enforce legal/licensing compliance frameworks for training data. • Acceptable Use Enforced: Prohibit unauthorized SaaS platforms and prompt sharing.	• C: Standardize SaaS use guidelines • I: Baseline vendor risk templates. • T: Set IP/licensing mandates.
Integration	• Identity Alignment: Map AI service accounts and user-delegated extensions directly to enterprise open directories. • Boundary Hardening: Treat all dynamic RAG pipelines and vector database inputs as untrusted.	• Directory-Based ACLs: Enforce corporate identity permissions to ensure least privilege across models. • Ingestion Hardening: Deploy filters to block indirect prompt injection via unstructured data. • Model Registry MFA: Enforce strict read-write isolation and MFA on training pipelines and repositories.	• C: Map platform logins to enterprise IAM. • I: Harden dynamic RAG corpora. • T: Secure model registries and weights.
Operationalization	• Inbound/Outbound Inspection: Deploy dedicated API gateways (e.g., Model Armor). • Supply Chain Visibility: Automate security cataloging across all models and API endpoints.	• Bidirectional Filtering: Scan ingress prompts for injections and egress outputs with network DLP to block PII/code leaks. • AI SBOM Generation: Integrate SCA tools into CI/CD pipelines to map libraries, APIs, and model files.	• C: Egress DLP checks. • I: Ingress semantic firewalls • T: Cryptographic model verification.
Continuous Operations	• Autonomous Guardrails: Enforce cost boundaries and performance limits on self-correcting agents. • Enterprise AI Literacy: Roll out continuous, role-based security training.	• Financial Circuit Breakers: Implement bounded recursion limits and project-level cost thresholds • Runtime Metrics Rollbacks: Revert configurations if active safety thresholds fall. • EDR/SIEM Calibration: Monitor agent telemetry	• C: SaaS safety training • I: Logic drift telemetry alerts. • T: Model safety rollbacks.

The following case study demonstrates how an absence of robust **governance FOR and OF AI** can lead to runaway execution loops and rapid cost inflation.

Case study 6: “Denial-of-Wallet” using rogue reasoning loop

A **global enterprise financial services** provider deployed an agent designed to reconcile accounting ledger anomalies, granting it direct read/write access to internal billing databases. When a corrupted, null value broke its formatting tool, the agent entered an unconstrained, recursive reasoning loop to brute force a fix. In under an hour it generated over 15,000 high-frequency, high-cost reasoning API calls, **triggering a sudden ~\$50,000 cloud-billing spike** and causing severe local database locking that halted active business transactions.

Defensive controls: To prevent rogue execution loops, organizations should define identity and access management and role-based access controls for Agents, cost-cap thresholds, and observability and monitoring requirements (governance FOR AI). Organizations should also define guardrails for agent autonomy. This includes implementing automated financial circuit breakers designed to halt agent operations after a set threshold of consecutive task failures. Additionally, [setting strict, real-time operational guardrails](#) (financial caps, bounded recursion limits and rate-limits) at the Service ID and project-level (governance OF AI) ensures runaway API calls and infinite self-correction loops are throttled before spiraling into severe financial costs or operational outages.

Challenges from the frontlines

Rapidly deploying AI into production introduces new security, operational, and architectural challenges.

Consequently, organizations struggle to apply deterministic policies to probabilistic model behaviors, creating blind spots across model provenance, unvetted pipeline extensions, and unauthorized runtime execution.

Mandiant has observed several key challenges security teams are facing on the front lines when adapting traditional defenses to modern AI pipelines:

1- Governance and lifecycle gaps: Outdated policies, standard SDLC exclusions, and the lack of internal SBOMs for proprietary tools.

2- Supply chain and model integrity: A broken chain of custody for open-weight assets, vulnerable model loaders (RCE risks), and public corpus contamination.

3- Access and ingestion vulnerabilities: Indirect prompt injection using RAG, missing metadata tagging defaults, and least-privilege service account violations.

Mandiant has also observed an increase in adoption of open weight models by organizations for specialized use cases. This can result in challenges as traditional security controls, governance models, and vulnerability management practices were not built for open-weight models. Open weight models shift the infrastructure and artifact security burden to the user. With open weight models, provenance details such as the original dataset, model architecture, and training code aren't always provided. To address these challenges, organizations should develop a security review process for open-weight models that includes cryptographic hash verification, a benchmarking process, bias testing, and vulnerability assessment for components (such as transformers). In

addition, organizations should ensure security for the hosting environment, data pipeline security for model fine-tuning, and monitoring for model drift.

AI guardrails and technical architecture

Through Mandiant's frontline AI and agentic security assessments, we frequently see that the enthusiasm for adopting AI can sometimes outpace the deployment of corresponding security guardrails. The threat landscape has evolved past simple, direct prompt injection, where a user types a misleading or malicious prompt, to **indirect prompt injection**. This occurs when the system architecture implicitly trusts data pulled from external pipelines, such as data from public web pages, customer emails, or user-uploaded documents stored in a vector database. The following case studies illustrate the risks associated with indirect prompt injection attacks and the absence of robust safeguards in AI deployments.

Case study 7: Customer service agent using untrusted input

Mandiant evaluated a **technology organization** that deployed a public-facing AI customer service agent designed to streamline user support. To answer customer queries, the agent relied on a Retrieval-Augmented Generation (RAG) knowledge base. The RAG pipeline ingested unstructured data from multiple sources, including community forum comments, customer service tickets, and data feeds provided by trusted third parties.

While the customer had strong controls around the vector database, they did not consider **indirect prompt injection**, which can stem from embedding hidden, malicious instructions inside a public forum comment and a customer support ticket. When the RAG agent retrieved these entries to resolve legitimate user inquiries, the model could interpret the embedded instructions as system-level commands. This could potentially hijack the model's reasoning, causing it to exfiltrate sensitive data from other customers' support tickets and expose PII directly in its response stream to unauthorized external users.

Defensive controls: Organizations should treat all retrieved RAG data, including community forums, support tickets, and external partner feeds as untrusted input. To mitigate indirect prompt injection, security teams should implement a defense-in-depth architecture that combines data segregation with runtime guardrails. Access controls, such as role-based, or tenant-isolated vector indexing, ensure that public agents cannot query sensitive cross-tenant data. Incorporating context isolation techniques, least-privilege tool execution, and egress data loss prevention (DLP) controls provide assurance that untrusted content cannot alter model reasoning or exfiltrate sensitive data in the response stream.

Incoming context streams should pass through dedicated security inspection layers (such as [Model Armor](#) or semantic firewalls) to strip prompt injections before reaching the model's context window. Semantic firewalls should also scan generated responses for sensitive data, PII, and credentials prior to rendering output to end users, preventing data leakage.

Securing the AI SDLC and pipeline supply chain

To prevent supply chain poisoning and compromised open-source dependencies, organizations need to **extend traditional secure software development lifecycle (S-SDLC) practices to their AI engineering pipelines**. An

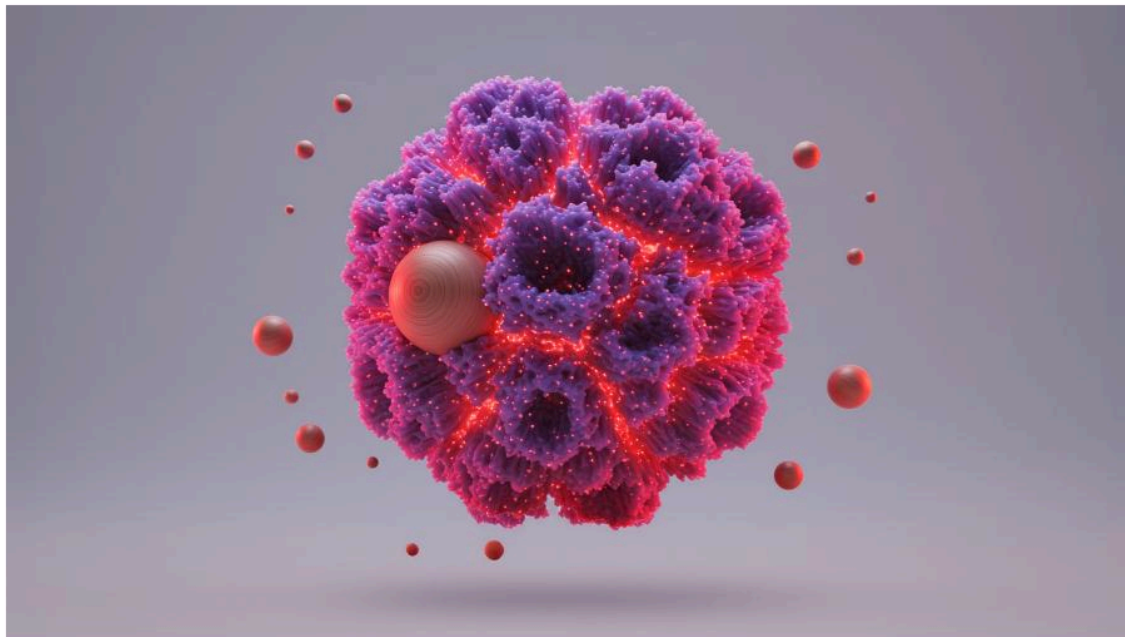
emerging observation is that some organizations are also fine-tuning self-hosted open-weight AI models for specific use cases. This adds a different dimension to existing SDLC processes and supply chain risk for organizations. Threat actors are aware of this shift and are targeting the SDLC to compromise enterprises. Beyond compromising popular AI and LLM libraries across open-source ecosystems like [LiteLLM](#) (March 2026), threat actors are uploading malicious components directly onto community platforms.

While adhering to Secure SDLC (S-SDLC) best practices is a fundamental baseline, organizations should further reduce blind spots and secure third-party components by embedding two foundational technical controls directly into their development lifecycle.

Firstly, **dynamic AI inventorying**. Maintain a centralized real-time catalog of AI use cases, associated risks, component applications, and cloud-hosted model endpoints. Secondly, **automated AI SBOM tracking**. Integrate Software Composition Analysis (SCA) tools directly into the CI/CD pipeline to automatically generate a SBOM for every build, while simultaneously deploying agentless compliance scanners to continuously discover active runtime software, model versions, and hardware configurations. By unifying automated pipeline tracking with real-time infrastructure discovery, security teams can establish a continuous, verifiable chain of custody for all AI assets from code to production.

To help defenders stay ahead of rapidly evolving adversarial AI, Mandiant has demonstrated the effectiveness of an [AI-Enabled Source Code Review](#) harness built on Gemini. By pairing a custom harness that automates and scales source code security reviews with a sequential pipeline of highly specialized AI agents, including threat modeling, entry point discovery, and taint analysis agents, our consultants can scale code reviews and focus on uncovering complex customized logic flaws and vulnerabilities. This point-in-time harness can also be used alongside [CodeMender's](#) ongoing scanning capabilities to create a multi-layered defense strategy.

In developing and delivering this new service, Mandiant consultants have already delivered measurable real-world results. By combining Gemini-enabled analysis with deep consulting expertise, we have helped clients identify and prioritize tens of thousands of code security findings, streamlining remediation, and accelerating risk reduction. This work has led to many assigned CVEs for zero-day vulnerabilities, including several in open-source software, with more currently in active disclosure. During a recent incident response investigation, the harness discovered **over 100 true-positive critical vulnerabilities in just two days**.



Modern defense architecture

Hardening agent identities

Deploying autonomous AI agents fundamentally alters the enterprise threat landscape by obfuscating attribution, authority and execution boundaries. Unlike traditional deterministic software, agentic systems act on behalf of users, reason dynamically, and invoke external tools. In these environments, identity becomes fluid, creating ambiguity in attribution and authority, that traditional authentication systems cannot fully address. Securing these systems requires transitioning away from traditional service accounts toward dynamic, cryptographically bound lifecycle-integrated identities (such as [Agent Identity](#)).

This transition is critical to mitigating the [lethal trifecta](#) that occurs when an agent is granted a combination of three critical capabilities; direct access to private data, exposure to untrusted content and the ability to externally communicate. When an agent operates at the intersection of these three factors, a threat actor can leverage both direct and indirect prompt injection to force the autonomous exfiltration of sensitive data to the threat actor's infrastructure.

Establishing structural resilience against these types of threats requires, replacing default service accounts with a defense-in-depth architecture across five critical pillars:

1- Hyper-segmented identity perimeters: Decouple agent's compute identities from data access layers using Workload Identity Federation, contextual IAM and workload bound [SPIFFE cryptographic identities](#) to eliminate long-lived keys. This baseline eliminates persistent credential exfiltration risks across cloud environments. (Case study 2)

2- Dual-execution authority: Prevent lateral escalation by differentiating headless background tasks from user-delegated actions. Use three-legged OAuth for synchronous workflows and require explicit human-in-the-loop re-

authorization for high-risk bulk operations. (Case study 3)

3- Data, tool, and operational governance: Control tool invocations, resource consumption, and data access (RAG) boundaries to prevent prompt injection, database poisoning, and supply chain compromises. Enforce user-context access controls on vector databases, sanitize data pipelines, and validate AI-recommended dependencies against cryptographic allowlists. To arrest non-deterministic logic failures and runaway billing loops, implement automated financial circuit breakers alongside strict API rate-limits and bounded recursion constraints. (Case studies 6 and 7)

4- Active defense and multi-layered telemetry: Enclose agent workloads within hardened network perimeters (such as [VPC-SC](#)) and deploy semantic firewalls like [Model Armor](#) alongside continuous behavioral telemetry across both model prompt streams and endpoint runtimes. Calibrate monitoring tools to detect indirect prompt injections, rogue reasoning loops, and host-level anomalies. This unified observability helps arrest data exfiltration, stealthy infrastructure pivoting and polymorphic malware evasion in real time. (Case studies 4 and 5)

5- Privileged developer sessions: Treat AI coding assistants and MCP servers as privileged sessions and isolate local credentials using just-in-time secrets management and by enforcing cryptographic integrity checks on all CLI hooks and plugins. This defends against local assistant hijacking, workstation credential harvesting, remote code execution and supply chain exploits. (Case studies 1 and 3)

Contextual analytics and the agentic SOC

In an agentic ecosystem, the expanded attack surface includes **Model Context Protocol (MCP) servers**, third-party APIs, and the dynamic agent instructions. Because adversarial misuse of AI agents can operate at machine-speed, protecting modern AI ecosystems requires security operations to bridge the gap between passive compliance trace logging and active, real-time monitoring. To achieve this, defenders should calibrate SIEM and EDR platforms to ingest rich telemetry (token usage metadata, cross-application API calls, agent application and harness logs, and agent access logs to sensitive assets).

Specifically, monitoring for 'first-time' access to sensitive repositories can serve as a high-fidelity signal, flagging potential internal reconnaissance that might otherwise be masked by seemingly legitimate tool calls. Mapping **behavioral indicators of activity (IOAs)** to sanctioned use cases and integrating Agentic AI with user and entity behavior analytics (UEBA), help overcome the limitations of human context windows to analyze massive long term datasets, flag real-time token anomalies, and rapidly contain subtle behavioral drifts associated with lateral movement or exfiltration.

To successfully deploy and operationalize this telemetry-driven strategy, security teams should focus on a three-stage life cycle:

1- Establish the baseline: consolidate high-fidelity telemetry over a 30-to-90-day window. Use dynamic standard deviation thresholds to automatically accommodate predictable operational spikes.

2- Automate triage and safeguards: Configure agents to automatically gather host status, directory groupings, and recent IP history, immediately upon detecting behavioral drift, before escalating to a human. Strictly isolate agent permissions to read-only access across critical telemetry systems.

- If high-fidelity IOAs signal that a guardrail has been bypassed, pre-authorized responders should **activate an emergency credential revocation** to instantly invalidate the agent's service identities and OAuth tokens using a Secret Manager.

3- Refine through feedback: Implement simple 'thumbs-up/thumbs-down' feedback mechanisms for security analysts to provide continuous feedback, refine the AI's detection accuracy, and prevent baseline drift over time.

Case Study 8: Disrupting a stealthy espionage campaign

Behavior-based detection successfully [thwarted a global cyber espionage campaign](#) (threat actor DARK CASTLE, [formerly tracked as UNC2814](#)) **targeting global telecommunications providers and government organizations**. The threat actors used a novel backdoor designed to blend in with normal network activity, by abusing legitimate cloud-based productivity spreadsheets for their command-and-control (C2) communications. When the security system flagged a single, low-frequency anomalous command execution on an endpoint, the defense team's agentic AI triage pipeline immediately escalated the alert. Typically, this type of alert would have been classified by a human analyst as having low criticality.

Defensive controls: By autonomously correlating this isolated endpoint anomaly with subtle outbound traffic signals, the agentic AI rapidly constructed a complete, unified attack timeline. This rapid detection allowed security engineers to quickly trace the actor's lateral movement (including SSH use and privilege escalation), completely severing the attacker's persistent access and neutralizing the threat before it could spread across other target environments.

FinOps for security operations

While deploying autonomous and agentic AI capabilities in the SOC delivers machine-speed defense, unleashing open-ended, autonomous capabilities alters the SOC's economic model. [Implementing FinOps is a technical necessity](#); categorizing workloads by security value versus token consumption ensures that expensive compute is reserved strictly for high-impact defensive actions.

Managing token economics requires implementing a tiered model architecture that aligns analytical complexity with appropriate model capacity. High-throughput, structured workloads (initial alert parsing, entity extraction, IOC correlation) are routed to lightweight, task-specific models, while larger frontier reasoning engines are reserved for multi-stage incident synthesis and threat hunting. We've recently introduced [Gemini 3.8 Flash Cyber](#), our most capable cybersecurity model with frontier-level performance in vulnerability detection and automated patching. To further optimize cost predictability at scale, security teams are increasingly adopting open-weight models (for example, Gemma). This strategy prevents linear token cost growth across continuous, telemetry-heavy monitoring pipelines while preserving data privacy and operational autonomy. Keeping routine baseline tasks (such as known-bad IP blocking) on lower-cost SIEM infrastructure aligns AI spend with security value.

Conclusion

AI systems are neither inherently unsecurable nor self-defending. They are complex, highly integrated software architectures that demand rigorous identity boundaries, continuous runtime observability, and rapid containment

mechanisms.

Resilience in 2026 is defined in part, by how effectively an enterprise governs and secures non-deterministic AI systems at scale. Organizations cannot rely on written policies or static perimeters alone to govern systems capable of autonomous execution. True resilience requires embedding technical guardrails into the software pipeline, monitoring agent behavior in real time, and empowering security teams to neutralize compromised workflows before lateral impact occurs.

Revisit our 2025 year in review, covering adversarial use of AI, shadow AI governance, red teaming for AI, and AI-powered defense in the previous [AI risk and resilience report](#)

Source: <https://cloud.google.com/security/resources/ai-risk-and-resilience-2026>