MCP SERVER | NO CODE | CLOUD HOSTED

GitGuardian MCP

Automate Breach Detection and Incident Response.

GitGuardian connects your workspace to any AI agent, letting you manage secret leaks and audit security without leaving your flow. Use it to scan code snippets, list active incidents, deploy decoy credentials (honeytokens), and check compliance logs instantly via natural conversation.

F Quality Score 3.6/100

secret-detection | vulnerability-scanning | incident-response | cybersecurity | honeytokens | code-security

MAINTAINED BY VINKIUS CLOUD FOR AI AGENTS · JUNE 2026
vinkius.com/mcp/gitguardian

SCAN TO VIEW

The infrastructure that powers AI agents in the real world.

Vinkius connects AI to the world's software through secure, enterprise-grade infrastructure — enabling real-world execution at scale, built on the Model Context Protocol (MCP).

Your AI Connections Run Through Vinkius Cloud

The world's largest managed MCP catalog

Vinkius is the cloud infrastructure where AI agents connect to the software your business already runs. We handle the hosting, the security, the credentials, the uptime — you get agents that actually do things.

We operate the world's largest managed MCP catalog. Major SaaS platforms, CRMs, databases, and cloud providers — running, monitored, production-ready. This MCP server is hosted and maintained by the Vinkius Cloud for AI Agents.

> *The agent doesn't manage credentials, doesn't manage uptime, doesn't manage security. Vinkius does.*
>
> — Architecture principle

Four Pillars of the Vinkius Runtime

01 — Security by design

Credentials stay encrypted at rest via AES-256. The AI agent never touches raw keys — they're injected into a sandboxed V8 isolate at runtime. Actions are logged, and connections have an emergency kill switch.

02 — Built on MCP Fusion

This MCP server was built with **MCP Fusion**, the open-source framework (Apache 2.0) that powers the entire Vinkius catalog. Schema-as-firewall strips undeclared fields, compiled PII redaction runs at zero overhead, and cryptographic lockfiles produce git-diffable audit trails.

03 — Deterministic observability

Eight immutable metrics per endpoint: request volume, p95 latency, error rate, active connections, cost attribution. A live payload feed logs every tool call with mutation detection.

04 — Autonomous operations

Servers are deployed, monitored, and patched autonomously. New capabilities and security patches ship weekly. Zero-downtime deployments ensure continuous availability across all managed MCP servers.

AES-256	Ed25519	24h TTL	V8 Isolate
Encryption at rest	PKI vault signatures	Ephemeral session keys	Sandboxed execution

One Token. Instant Access.

Every MCP server on Vinkius is accessed through a **Connection Token**. Tokens are generated in the cloud dashboard and produce a unique MCP endpoint URL. Paste this URL into any MCP-compatible client — no SDK required.

A single token can serve **multiple AI clients simultaneously**, or you can issue separate tokens per client for granular access control. Each token tracks its own request count, last activity timestamp, and can be individually enabled or revoked.

MCP ENDPOINT

```
https://edge.vinkius.com/{token}/mcp
```

Claude · Cursor · VS Code · Windsurf · Grok · Gemini

Security Is the Architecture

Security in Vinkius is not a feature — it's the foundation of the runtime. The gateway enforces multiple independent protection layers between AI agents and third-party APIs.

01 — Ed25519 PKI Vault

Every workspace has an Ed25519 Master Key. Session keys are generated ephemerally (24h TTL) and signed by the Master Key. Credentials never leave the vault boundary.

02 — V8 Isolate Sandboxing

Tool code runs inside isolated-vm V8 isolates with 64 MB memory caps and per-request timeouts. No filesystem access, no network access except through the SSRF-guarded fetch bridge.

03 — SSRF Guard

All outbound HTTP requests are DNS-resolved and validated before execution. Private IP ranges (10.x, 172.16-31.x, 192.168.x, AWS metadata 169.254.x) are blocked at the network layer.

04 — DLP & PII Redaction

A ResponseGuard pipeline intercepts every tool response. Configurable redaction patterns strip sensitive fields (emails, SSNs, card numbers) before data reaches the AI agent.

05 — Cryptographic Audit Trail

Every request is signed into a SHA-256 hash chain with Ed25519 signatures. Events form a tamper-proof, SIEM-exportable forensic record.

06 — Honeytoken Trap System

Phantom credentials are injected into isolated environments. If a honeytoken is used outside Vinkius infrastructure, the server is quarantined instantly.

Emergency Kill Switch

EU AI Act Art. 14(1) Compliant

The kill switch is an **emergency halt** mechanism — not a simple toggle. When triggered, it executes three actions atomically:

01 — Server deactivated

The MCP server is immediately taken offline across the entire cluster.

02 — All tokens revoked

Every connection token is invalidated. Total lockout — reconnection blocked until new tokens are issued.

03 — WebSocket connections killed

Active connections terminated via Redis pubsub broadcast. Propagates to every runtime node in the cluster.

Full Visibility. Zero Guesswork.

The Vinkius cloud dashboard includes a full MCP Governance suite — real-time analytics and security controls for production AI operations.

Control Plane

KPI dashboard with request volume, latency, success rate, token consumption, and AI-generated operational briefings.

Agent Activity

Which AI clients are connecting, how often, and what they're doing — real-time session tracking.

FinOps

Cost tracking per tool, payload compression savings, budget optimization signals, and consumption trends.

Tool Health

Slowest and most error-prone tools, with actionable root-cause insights and performance baselines.

Firewall & DLP

PII redaction activity, sensitive data protection counters, and security event timeline.

Incident Log

Error trends, failure rates, status-code breakdowns, and forensic audit trail access.

> Get started at **cloud.vinkius.com** — connect your AI agent in under 60 seconds.

GitGuardian MCP

49 tools available
Cloud-hosted on Vinkius

This MCP lets you strengthen your organization's security by automating the detection of leaked secrets. You talk to your agent, and it handles the heavy lifting—scanning code for hardcoded API keys or AWS credentials before they cause a breach. If an incident is already active, you don't have to log into multiple dashboards; you can ask your agent to list secret incidents, assign them to specific team members, or even update their status right away. You can also build detection layers by creating and managing decoy honeytokens that flag unauthorized access attempts across your private infrastructure. Because Vinkius hosts this MCP, your agent gets instant access to all the security tools needed, allowing you to operate as a 24/7 Security Operations Center assistant directly from your IDE or terminal.

Core Capabilities

01 — Scan Code for Secrets

You feed the MCP code snippets or documents and it finds sensitive data like private keys, tokens, and passwords.

02 — Manage Breaches and Incidents

You can list active leaks, get full details on a specific secret incident, and resolve them when remediation is complete.

03 — Deploy Decoy Credentials

The MCP allows you to create honeytokens—fake credentials placed in your system that alert you if they are ever used by an unauthorized party.

04 — Audit Compliance and Activity

You retrieve detailed logs of workspace activity, track who did what, and ensure the environment meets security policy requirements.

05 — Organize Security Teams

The MCP helps you manage team memberships and API tokens to coordinate security efforts across different groups in your company.

One Click on Vinkius — From Prompt to Execution

Available at **vinkius.com/mcp/gitguardian** — connect your AI agent in three steps.

01 First, subscribe to this MCP and provide your specific GitGuardian API Key.

02 Next, tell your AI client what you want to check—for example, 'Scan the latest pull request for secrets,' or 'List all active honeytokens.'

03 Finally, your agent processes the request using the underlying tools, returning a clean summary of detected leaks, incident status, or audit results.

The bottom line is you get an automated security analyst that lives inside your existing workflow and doesn't require switching tabs.

Built For

This MCP is for Security Engineers, DevOps/SREs, and developers who are tired of manually logging into multiple dashboards just to check if a key was accidentally committed. It gives you a central point of control over your code's security posture.

DevOps Engineer

You use the MCP to run automated security audits, checking health status or listing audit logs across multiple environments without writing custom scripts.

Security Analyst

You ask your agent to list secret incidents and immediately assign them or resolve them after verifying the fix, speeding up incident response time dramatically.

Developer

Before committing code, you run a quick scan through the MCP to ensure no API keys or sensitive credentials accidentally slip into your repository.

What Changes When You Connect

01 You stop guessing about security. By using the `scan_content` tool, your agent instantly checks any code snippet you provide for sensitive data before it ever makes it into a commit.

02 Incident response is faster. Instead of manually checking dashboards, you ask to list secret incidents and then use `assign_secret_incident` or `update_secret_incident` to manage the fix status right in your chat interface.

03 You build better defenses with honeytokens. Running `create_honeytoken` lets you deploy fake keys across your infrastructure, and if they trigger an event (which you can list using `list_honeytoken_events`), you know exactly where an intruder is looking.

04 Compliance checks get automated. You can ask the MCP to run a full audit by listing all audit logs or checking the IP allowlist rules without logging into separate compliance portals.

05 Your team coordination improves. The MCP lets you manage teams via `create_team` and track who has access using tools like `list_members`, keeping your security operations organized.

Real-World Applications

Preventing Accidental Key Commits

A developer is about to push a new feature branch. Instead of running local checks, they ask their agent: 'Scan this file for secrets.' The agent uses `scan_content` and immediately flags an exposed Stripe key, allowing the developer to fix it before committing.

Responding to Suspected Breaches

A security analyst notices strange activity. They ask the agent to list all honeytokens events. The agent uses `list_honeytoken_events`, which shows that a decoy AWS key was used in an unexpected region, guiding the investigation immediately.

Auditing Team Access

A manager needs to know who has access rights across environments. They ask the agent to list all workspace members and then use `list_team_memberships` to verify if a departed employee still belongs to critical groups.

Maintaining Compliance Records

During an audit, you need proof of security controls. You ask the agent to list audit logs for the last quarter. The MCP uses `list_audit_logs` and provides a structured report showing all critical actions taken.

Patterns to Avoid

Treating it like a simple log viewer

✗ AVOID

Just asking the agent to 'show me what happened.' This only gives you passive data and doesn't help you fix or prevent things from happening again.

✓ INSTEAD

To actively manage an issue, first use `list_secret_incidents` to identify the leak, then use `assign_secret_incident` to hand off ownership, followed by `update_secret_incident` once it's resolved.

Ignoring decoy credentials

✗ AVOID

Seeing a honeytoken event pop up but dismissing it as 'just noise.' This is how real attackers operate; they test decoys first.

✓ INSTEAD

If you see an alert, use `list_honeytoken_sources` to pinpoint exactly where the unauthorized access originated. Then use `get_secret_incident` to determine if that location has other vulnerabilities.

Manual workflow updates

✗ AVOID

Having to jump between a ticketing system, GitHub dashboard, and GitGuardian web UI just to update status.

✓ INSTEAD

Use `update_secret_incident` directly in your chat interface. This single action records the change, updating both your ticket and the security record instantly.

The Right Fit

You need this MCP if your primary pain point is managing leaked credentials or ensuring code remains clean of secrets. If you just want to track general system usage (e.g., who logged in), a basic log viewer might suffice. But when the threat involves highly sensitive data like API keys, access tokens, or private project identifiers, GitGuardian is necessary. Use it if your workflow requires actions like creating decoys (`create_honeytoken`), triaging incidents (`list_secret_incidents`), and automating compliance checks across multiple systems. Don't use this MCP just to list teams; for simple user management, a dedicated directory tool works better. However, if you need the audit log alongside team structure details, this MCP connects all those dots.

The Security Dashboard Maze

Today, finding out what went wrong with your code is a multi-tab nightmare. You have to click over to the GitGuardian dashboard, filter by date, manually review incident summaries, and then copy details into a separate ticketing system just to assign ownership or update the status. It’s slow, it's painful, and you spend more time clicking than securing.

With this MCP, that entire manual process collapses into a single conversation. You simply ask your agent about the leaks. It retrieves all necessary information—from listing secret incidents to checking audit logs—and presents it instantly, letting you take action without leaving your current workflow.

Incident Management and Audit Visibility

Manual incident management requires people to remember which keys were leaked, who owns the remediation plan, and what steps have already been taken. You waste time cross-referencing `list_secret_incidents` data with team directories and change logs.

Now you can ask your agent to handle it all. It pulls up the incident details, shows the responsible team via `list_team_memberships`, and lets you confirm remediation status using `resolve_secret_incident`. The entire security lifecycle moves from a manual series of clicks into one conversation.

GitGuardian: 49 Tools for Code Security

These tools allow you to perform every level of security operations—from listing team members to scanning content for secrets—all through natural language conversation.

#	TOOL	DESCRIPTION
01	assign_secret_incident	This tool lets you assign ownership of an existing secret leak incident to a specific team member.
02	bulk_prefix_lookup	It performs a bulk lookup for common honeytoken hashes, helping confirm if a decoy credential was triggered.
03	create_custom_tag	You create specific tags to categorize or label security findings within your workspace.
04	create_honeytoken_note	This tool allows you to attach contextual notes directly to a honeytoken for documentation purposes.
05	create_honeytoken	You deploy new decoy credentials (honeytokens) into your system, increasing detection coverage.
06	create_honeytoken_with_context	This lets you create a honeytoken and simultaneously add specific contextual information to it.
07	create_team	You establish new teams within your GitGuardian account for grouping users with shared security responsibilities.
08	delete_custom_tag	This removes a custom tag you previously created, cleaning up unnecessary labels.
09	delete_custom_tags_key	You delete an entire key of custom tags when they are no longer needed.
10	get_custom_tag	This retrieves the details for a specific, existing custom tag by its name or ID.
11	get_health	You check the overall API health status of your connected GitGuardian account to ensure proper connection.
12	get_honeytoken	This retrieves all stored details for a single honeytoken, letting you review its setup and usage history.
13	get_quotas	You view an overview of your current API usage quotas to prevent service interruptions.

#	TOOL	DESCRIPTION
14	`get_secret_incident`	This tool retrieves all historical and current details related to a specific secret leak incident.
15	`get_self_api_token`	You pull the full details of the API token currently being used by your agent client.
16	`ignore_secret_incident`	If a leak is false positive or benign, you can mark it as ignored to clear up unnecessary alerts.
17	`list_api_tokens`	You get a list of all API tokens associated with your workspace for auditing purposes.
18	`list_audit_log_event_names`	This lists every type of event that can be tracked and audited within your workspace history.
19	`list_audit_logs`	You view a comprehensive list of all activity logs, showing who did what and when in the workspace.
20	`list_custom_tags`	This retrieves an overview of every custom tag you have set up for organization.
21	`list_health_check_history`	You view a record of past health checks to track stability over time for a specific instance.
22	`list_health_checks`	This lists the current and recent health check records available for your monitored environment.
23	`list_honeytoken_events`	You retrieve a list of all events triggered by any honeytoken, showing detection activity.
24	`list_honeytoken_notes`	This shows you all the documentation notes that have been attached to your honeytokens.
25	`list_honeytoken_sources`	You see a list of sources where any given honeytoken has appeared, pinpointing potential intrusion points.
26	`list_honeytokens`	This provides an overview and list of all currently active decoy credentials (honeytokens).
27	`list_ip_allowlist`	You view the current rules defining which IP addresses are permitted access to your system.
28	`list_ips`	This lists all official and monitored IP address ranges belonging to GitGuardian's infrastructure.
29	`list_members`	You view a roster of all user accounts who have access to the workspace.

#	TOOL	DESCRIPTION
30	list_scim_groups	This lists groups that are synced or managed via SCIM protocols, helping with identity management.
31	list_scim_users	You view a list of users who have been imported into the workspace using SCIM standards.
32	list_secret_incidents	This retrieves a comprehensive list and summary of all detected secret leaks in your entire codebase or repository.
33	list_sources	You view every source type (e.g., GitHub, GitLab) that is currently connected and being monitored for secrets.
34	list_team_memberships	This shows which specific users belong to which security teams within your organization.
35	list_teams	You get an overview and list of all defined security teams in the workspace.
36	multiscan_content	This tool allows you to scan multiple large files or documents simultaneously for patterns indicating secrets.
37	reset_honeytoken	If a decoy credential is compromised, this resets it so that you can redeploy a fresh copy immediately.
38	resolve_secret_incident	When a security issue has been fixed and verified, you use this to formally close out the incident record.
39	revoke_honeytoken	You deactivate a honeytoken, preventing it from being triggered or reported on further.
40	revoke_self_api_token	This immediately cancels and revokes the specific API token your agent is currently using for connectivity.
41	scan_and_create_incidents	You run a scan on new content, and if secrets are found, this automatically generates official incident records.
42	scan_content	This scans a single provided piece of content or code snippet to immediately check for any sensitive data patterns.
43	trigger_health_check	You force the system to run an immediate health check on your monitored environment, verifying uptime and connectivity status.
44	trigger_source_scans	This manually kicks off scans across all connected sources (like repositories) to catch any recent changes that might contain secrets.
45	update_custom_tag_full	You perform a complete, full replacement of the details associated with an existing custom tag.

#	TOOL	DESCRIPTION
46	update_custom_tag_partial	This modifies only specific fields of an existing custom tag without affecting its other metadata.
47	update_custom_tags_key	You update the key definition for a group of custom tags when their purpose changes.
48	update_honeytoken	If an existing decoy credential needs modification (like changing its name), this updates its details while keeping it active.
49	update_secret_incident	You modify the status or assigned personnel of a secret incident after initial detection.

See It in Action

Real prompts you can use once this MCP is connected to your AI agent through Vinkius Cloud.

U List the most recent secret incidents in my workspace.

I've retrieved the latest incidents. There are 3 active leaks: a 'GitHub Personal Access Token' in repository 'web-app' and two 'AWS Keys' in 'legacy-service'. Would you like the details for the GitHub token leak?

U Create a new AWS honeytoken named 'Internal-Dev-Decoy'.

Successfully created the honeytoken 'Internal-Dev-Decoy'. The AWS Access Key ID is `AKIA...` and the Secret Key is provided in the metadata. You can now place this in your environment to detect intruders.

U Scan this code for secrets: 'const apiKey = "sk_live_51MzX..."'

Scan complete. I detected a 'Stripe Live Secret Key' in the provided string. This is a high-severity finding. I recommend rotating this key immediately and removing it from your source code.

Frequently Asked Questions

01 **How do I use GitGuardian MCP to find leaked API keys?**

You ask your agent to scan specific code snippets using the `scan_content` tool. It immediately checks that content against known patterns for secrets and reports any findings, telling you which key was exposed.

02 **Can GitGuardian MCP manage my team's security roles?**

Yes, you use tools like `list_teams` or `list_members` to see who is in the system. You can then use `assign_secret_incident` to assign ownership of a breach to specific team members.

03 **What are honeytokens and how does GitGuardian MCP help?**

Honeytokens are fake credentials that act as tripwires. The MCP lets you deploy them using `create_honeytoken`. If they get used, the system alerts you via `list_honeytoken_events`, showing exactly where an intruder went.

04 **Is GitGuardian MCP better than just looking at audit logs?**

Theoretically, yes. While you can use `list_audit_logs` to see general activity, this MCP connects that log data directly to specific secret incidents and team responsibilities, giving context.

05 **How do I clean up old or false positive leaks with GitGuardian MCP?**

First, you check the details using `get_secret_incident`. Once confirmed as benign or fixed, you use `ignore_secret_incident` to mark it in the system, keeping your active incident list clean.

Go Live in 60 Seconds

Get your connection token from **cloud.vinkius.com**, then paste the endpoint URL into any MCP-compatible client.

YOUR MCP ENDPOINT

```
https://edge.vinkius.com/[TOKEN]/mcp
```

CLIENT	WHERE TO CONFIGURE
Claude AI	Profile → Customize → Connectors → "+" → Add custom connector → Paste endpoint
Cursor	Settings → Features → MCP Servers → "+ Add New MCP Server" → Type: SSE → Paste endpoint
VS Code	Ctrl/Cmd+Shift+P → "MCP: Add Server" → add `"gitguardian": { "url": "…" }`
Windsurf	MCP Settings → `mcp_settings.json` → Add endpoint URL
ChatGPT	Settings → Tools & plugins → Add MCP server → Paste endpoint
Gemini	Extensions → Add MCP Server → Paste endpoint URL

ASK AN AI ABOUT THIS

Let your preferred AI explain this MCP server

- Ask ChatGPT
- Ask Claude
- Ask Perplexity
- Ask Gemini
- Ask Grok

READY TO CONNECT

GitGuardian is live on Vinkius Cloud.

Get your connection token, paste it into your AI agent, and start building. No SDK. No deployment. Just results.

Start at cloud.vinkius.com →

vinkius.com · support@vinkius.com

INDEPENDENT PLATFORM DISCLAIMER

Vinkius is an independent platform and is not affiliated with, endorsed by, sponsored by, verified by, or otherwise authorized by GitGuardian. All third-party trademarks, logos, and brand names are the property of their respective owners. Their use in this document is strictly for informational purposes to identify service compatibility and interoperability.

DOCUMENT INFORMATION

Generated	June 2026
MCP Server	GitGuardian MCP
Server ID	`019e389f-238f-717f-8dba-ee0217ad21b1`
Platform	Vinkius Cloud for AI Agents
Endpoint	`https://edge.vinkius.com/{token}/mcp`

LICENSE & USAGE

This document is generated automatically by the Vinkius PDF Engine. Content reflects the MCP server configuration at the time of generation and may change as updates are deployed. For the most current information, visit vinkius.com/mcp/gitguardian.