Tackling Secret & Vault Sprawl with AI

GitGuardian + CyberArk MCP Servers in Action

A joint webinar — GitGuardian & CyberArk

Your Hosts Today

Mathieu Bellon	**Or Geisler**	**Dwayne McDaniel**
Product Manager	Software Engineer	Dev Advocate · *Moderator*
GitGuardian	CyberArk	GitGuardian

> *Questions? Drop them in the chat — we'll have a live Q&A at the end.*

What We'll Cover Today

1. **Live Demo** — Watch AI detect secrets and vault them automatically
2. **Why We Built This** — The rise of NHIs, secret sprawl, and vault sprawl
3. **The Bigger Picture** — Identity-based workloads vs. enterprise reality
4. **GitGuardian Platform & MCP** — Detection, remediation, honeytokens
5. **CyberArk Conjur & MCP** — Secrets management and vault integration
6. **Demo Deep Dive** — Step-by-step walkthrough of what happened
7. **Putting It All Together** — Key takeaways and Q&A

Live Demo

Watch the full workflow end-to-end

Then we'll break it all down

What You're About to See

A developer working in their IDE with **two MCP servers** running side by side:

- **GitGuardian MCP** — detects hardcoded secrets in real time
- **CyberArk Conjur MCP** — stores secrets securely in Conjur vault

The AI agent **orchestrates both** — no manual steps, no context switching.

> *Pay attention to the flow. We'll slow it down and explain every step after.*

End-to-End Workflow

```
  Developer (IDE)         GitGuardian MCP         CyberArk MCP
        │                        │                       │
        │  "Review my code"      │                       │
        │───────────────────────►│                       │
        │                        │  scan_secrets         │
        │  ⚠️ Found 3 secrets     │                       │
        │◄───────────────────────│                       │
        │                        │                       │
        │  "Help me fix these"   │                       │
        │───────────────────────►│  remediation plan     │
        │                        │──────────────────────►│
        │                        │                       │  store in vault
        │                        │◄──────────────────────│  return references
        │  ✅ Secrets vaulted     │                       │
        │◄───────────────────────│                       │
        │  Code updated with vault references            │
```

Why We Built This

The story of NHI rise and secret sprawl

The Rise of Non-Human Identities

The number of **non-human identities** (NHIs) is exploding:

- API keys, service accounts, tokens, certificates, OAuth apps
- NHIs now outnumber human identities **50:1** in most enterprises
- Every microservice, CI/CD pipeline, and cloud resource needs credentials
- **Each NHI is a potential attack vector** if not properly managed

> *The average enterprise manages over 10,000 NHIs — most with no lifecycle management.*

Secrets Are Everywhere

Hardcoded secrets are **one of the most common security vulnerabilities**:

- API keys, tokens, passwords committed to repos
- Detected **too late** — often after push
- Traditional scanners run in CI/CD — **feedback loop is slow**
- Developers must context-switch to security dashboards

> ***70%+*** *of leaked secrets remain valid and exploitable days after exposure.*

What if your AI coding assistant could catch secrets **before you commit**?

Vaults Are the Right Answer — But...

Vaults are the **correct** answer. **The problem? Vault sprawl.**

- Multiple vaults across teams, clouds, and environments
- Developers don't know **which vault to use** or **how to use it**
- Each vault has its own API, CLI, and auth flow

The ideal	The reality
All secrets in vaults	Scattered across code, configs, .env
Clear ownership	Nobody knows who owns what
Automated rotation	Manual — or none at all
One vault	**3–5 vaults** per enterprise

The Missing Link: Detection to Remediation

Today's workflow is **broken**:

```
Developer commits secret
       ▼
Scanner detects it (CI/CD — hours later)
       ▼
Alert fires → dashboard → ticket
       ▼
Developer context-switches
       ▼
Manually rotates + vaults the secret  →  30-60 min of toil
```

What if AI could bridge this gap — from detection to vault — in seconds?

The Bigger Picture

Identity-based workloads and enterprise reality

Where We Should Be Heading

The industry is moving toward **identity-based workloads**:

- Workload identity federation (no more long-lived secrets)
- Just-in-time credential issuance
- Zero standing privileges
- Every workload authenticated by identity, not by secret

But the messy reality? This transformation takes **years** in the enterprise.

In the meantime, teams need practical tools to manage the **millions of secrets that exist today** — scattered across repos, configs, and environments.

> *Catch our previous webinar for a deeper dive on this topic — link in the chat.*

GitGuardian

Platform overview & MCP architecture

GitGuardian at a Glance

The #1 secrets detection platform, trusted by 600K+ developers.

- **500+ secret detectors** — API keys, tokens, passwords, certificates
- **Monitors code everywhere** — GitHub, GitLab, Bitbucket, CI/CD, Docker images
- **Real-time scanning** — pre-commit, pre-push, and in CI/CD pipelines
- **Incident management** — track, assign, and remediate from one dashboard
- **Honeytokens** — plant decoy credentials to detect intrusions

Works across the full SDLC — from IDE to production.

Quick MCP Refresher

Model Context Protocol — an open standard for connecting AI agents to external tools.

```
┌──────────────────┐          ┌──────────────────┐
│   MCP Client     │  JSON-RPC│   MCP Server      │
│ (AI App / Agent) │◄────────►│ (Tool / Service)  │
│  Claude, Cursor, │   2.0    │  GitGuardian,      │
│  Windsurf, etc.  │          │  CyberArk, etc.    │
└──────────────────┘          └──────────────────┘
```

- Agent **discovers** tools at runtime and **decides** which to call
- **Write once, use everywhere** — any MCP client can use any MCP server
- Maintained by the **Linux Foundation** (Agentic AI Foundation)

GitGuardian MCP Server

Real-time secrets detection directly in your IDE — wherever your agent lives.

Tool	What it does
`scan_secrets`	Scan code for **500+ secret types**
`list_incidents`	View secret incidents in your repo
`remediate_secret_incidents`	Guided remediation with **best practices**
`generate_honey_tokens`	Honeytokens for **breach detection**

```
{ "mcpServers": { "gitguardian": {
    "command": "pipx", "args": ["run", "ggshield", "mcp"] } } }
```

Open source — github.com/GitGuardian/gg-mcp

Built for Security

The GitGuardian MCP Server is built with security in mind:

- **Read-only by design** — minimizes risk, no destructive actions
- **Safe and supervised** — agent behavior is auditable
- **500+ detectors** — covers all major secret types
- **Works with your stack** — language and framework agnostic

> *"A new security primitive — proactive, context-aware security actions directly in the development environment."*
>
> *— Eric Fourrier, CEO GitGuardian*

CyberArk Conjur

Platform overview & MCP architecture

CyberArk Conjur at a Glance

Enterprise-grade secrets management for DevOps and cloud-native workloads.

- **Centralized secrets vault** — store, rotate, and manage credentials at scale
- **Policy-as-code** — access control defined in declarative policies
- **Dynamic secrets** — just-in-time credential generation
- **Integrations everywhere** — Kubernetes, Ansible, Terraform, Jenkins, CI/CD
- **Audit trail** — every secret access is logged and traceable

Conjur is the foundation for securing **non-human identities** in the enterprise.

CyberArk Conjur MCP Server

AI-driven secrets vaulting directly from your IDE.

The CyberArk MCP server lets an AI agent **store secrets in Conjur** — the developer never leaves their editor.

Tool	What it does
`store_secret`	Store a credential securely in Conjur vault
`retrieve_secret`	Fetch a secret by its vault reference
`list_secrets`	List available secrets and their policies
`rotate_secret`	Trigger rotation of a stored credential

The agent handles the vault API, auth, and policy — **the developer just says "fix it"**.

Two MCP Servers, One Workflow

GitGuardian and CyberArk MCP servers work **side by side**:

```
 AI Agent (IDE)       GitGuardian MCP       CyberArk Conjur MCP

┌───────────────┐    ┌─────────────────┐    ┌─────────────────┐
│ Orchestrates  │───►│  DETECT         │    │                 │
│ both servers  │    │  What secrets   │    │                 │
│ automatically │    │  are exposed?   │    │                 │
│               │    │  REMEDIATE      │───►│  VAULT          │
│               │    │  How to fix it? │    │  Store securely │
└───────────────┘    └─────────────────┘    └─────────────────┘
```

- **Lower cognitive load** — devs don't need to know vault APIs
- **Faster remediation** — from detection to vault in seconds
- **Consistent policy** — secrets land in the right vault with the right ACLs

Demo Deep Dive

Let's walk through each step

The Starting Point

The developer finds themselves in a **common situation**:

- Working on a feature, pulling in configs from various sources
- Hardcoded API keys, database passwords, cloud credentials in the code
- **They may not even realize** some of these are secrets

```
# config.py – a typical file with secret sprawl
AWS_ACCESS_KEY = "AKIA2E0A8F3B244C9986"
DB_PASSWORD = "super_secret_prod_password"
SLACK_WEBHOOK = "https://hooks.slack.com/services/T00/B00/xxxxx"
```

This is the starting point for most secret leaks.

Asking AI for Help

The developer knows **plaintext secrets are bad** — they ask the AI agent:

```
Developer: "Can you review my code for security issues
            and help me fix anything you find?"
```

The developer doesn't need to know:

- Which scanner to use or how to configure it
- Which vault to store secrets in
- What the vault API looks like

The AI agent figures all of that out using the MCP servers available to it.

GitGuardian MCP in Action

The agent calls `scan_secrets` and detects **3 hardcoded credentials**:

```
Agent: ⚠️ I found 3 secrets in config.py:
       1. AWS Access Key (line 2) — HIGH severity
       2. Database Password (line 3) — HIGH severity
       3. Slack Webhook URL (line 4) — MEDIUM severity
       Let me remediate these for you.
```

Without MCP: run CLI manually, read output, look up docs, file tickets.

With MCP — it happens in one natural language request.

CyberArk Conjur MCP in Action

The agent calls the CyberArk MCP server to **vault each secret**:

```
Agent: I'll store these secrets in Conjur vault:
       1. ✅ AWS key → conjur/prod/aws/access-key
       2. ✅ DB password → conjur/prod/db/password
       3. ✅ Slack webhook → conjur/prod/slack/webhook
       Updating your code to use vault references...
```

Without MCP: log into Conjur, understand policy structure, create paths/ACLs, manually replace secrets, test runtime access.

With MCP — the AI handles vault API, auth, policies, and code updates.

The Result

The code is now **clean and secure**:

```
# config.py - after AI remediation
import os
AWS_ACCESS_KEY = os.environ["CONJUR_AWS_ACCESS_KEY"]
DB_PASSWORD = os.environ["CONJUR_DB_PASSWORD"]
SLACK_WEBHOOK = os.environ["CONJUR_SLACK_WEBHOOK"]
```

- Secrets in **Conjur vault** with proper policies
- Code uses **environment variables** injected at runtime
- **No hardcoded secrets** — no risk of leaking to git

Total time: seconds. Without AI: 30-60 min per secret.

Putting It All Together

From detection to vault in seconds

The Complete Workflow

```
Developer (IDE)        GitGuardian MCP        CyberArk MCP
     │                        │                       │
     │  "Review my code"      │                       │
     │───────────────────────►│                       │
     │                        │  scan_secrets         │
     │   ⚠️ Found 3 secrets    │                       │
     │◄───────────────────────│                       │
     │                        │                       │
     │  "Help me fix these"   │                       │
     │───────────────────────►│  remediation plan     │
     │                        │──────────────────────►│
     │                        │                       │  store in vault
     │                        │◄──────────────────────│  return references
     │   ✅ Secrets vaulted    │                       │
     │◄───────────────────────│                       │
     │   Code updated with vault references           │
```

Why This Matters

Without MCP	With GitGuardian + CyberArk MCP
Secrets found in CI/CD (hours later)	Secrets found **while coding**
Manual triage and ticket creation	**AI-guided** remediation in IDE
Developer looks up vault docs	AI **knows** the vault API
30-60 min per secret to vault	**Seconds** per secret
Context switch to 3+ tools	**Stay in your editor**
Cognitive load on the developer	**AI handles the toil**

Find It. Fix It. Sustain It.

Find it. Detect exposed secrets and risky patterns with GitGuardian's 500+ detectors — in your IDE, in CI/CD, across all repos.

Fix it. AI uses CyberArk Conjur MCP to vault secrets with the right policies, automatically. From detection to remediation in seconds — no context switching.

Sustain it. Make it continuous. Monitor for regressions, enforce rotation, track NHI lifecycle — even as teams, repos, and vaults scale across the enterprise.

> *One workflow. Two MCP servers. A secure-by-default developer experience.*

Key Takeaways

1. **Secret & vault sprawl is real** — NHIs outnumber humans 50:1 and growing
2. **Find it** — GitGuardian detects secrets in your IDE, before they reach git
3. **Fix it** — CyberArk Conjur vaults them automatically via MCP, no context switching
4. **Sustain it** — continuous monitoring, rotation enforcement, and governance at scale
5. **MCP is the glue** — AI agents orchestrate both, so developers focus on code

Thank You

Questions?

gitguardian.com · cyberark.com/conjur · github.com/GitGuardian/gg-mcp · modelcontextprotocol.io