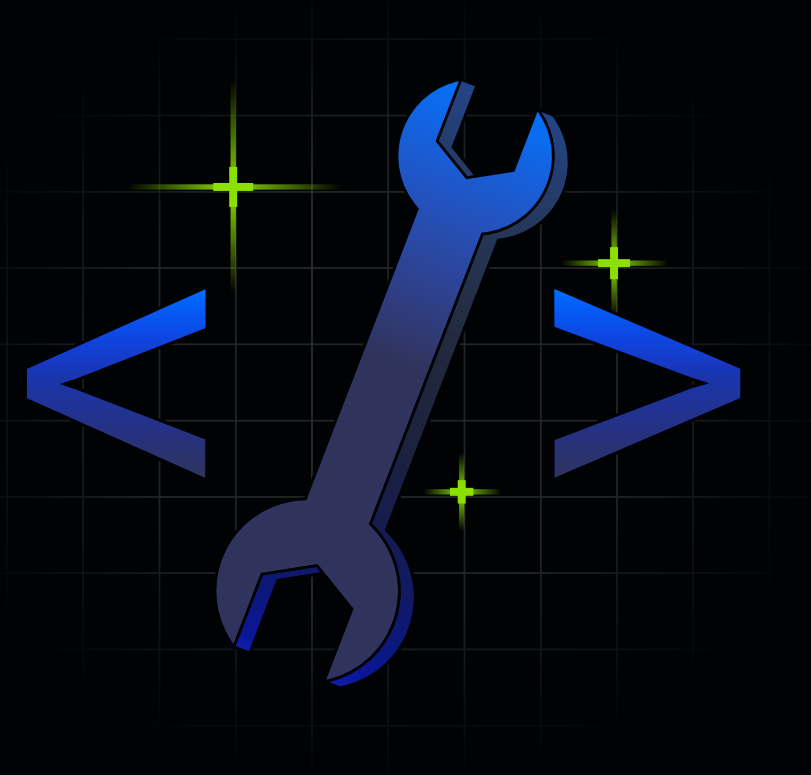
VARONIS

GUIDE TO DEV CYCLE DATA SECURITY

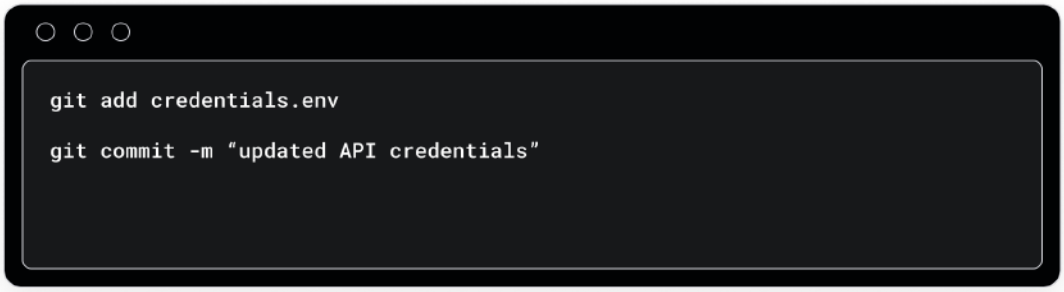
Don’t let your developer tools become attack vectors.



THE BLIND SPOT

The development process is a blind spot for security teams. Without the right security controls, sensitive data, secrets, and credentials can easily be put at risk.

API keys get hardcoded into repositories. Customer PII gets pasted into Jira tickets. Credentials show up in Confluence runbooks. Secrets persist in commit history long after everyone thinks they've been deleted. Code with embedded secrets, customer data, and internal configs gets pasted directly into AI assistant prompts with zero organizational visibility.



Stolen credentials alone are an initial attack vector in 22% of breaches, and the average cost of a credential-driven breach is $4.67 million (Source: IBM 2025). GitHub found that developers leaked 39M secrets in 2024 alone. As the pace of code being created increases, so does the amount of secrets being leaked.

Few security teams have visibility into the tools developers use to plan, build, and collaborate. As a result, developer tools become easy-to-miss vectors for data leakage and easy targets for attackers.

Native platform controls like GitHub secret scanning, Atlassian Guard, JFrog Xray, and Slack DLP were designed to protect individual platforms from external threats, not to gain visibility and control across the development cycle. These tools alone can’t answer critical questions, like What sensitive data do we have across our developer tool stack? Who can access it? Who should access it? Who has accessed it? Should it be here at all?

In this guide, we’ll show you how you can use Varonis to gain visibility into the development lifecycle — including the AI development lifecycle — and prevent data breaches and data loss.

ANATOMY OF A DEV DATA BREACH

In the past two years, some of the world's largest and most sophisticated organizations, including Cloudflare, Cisco, Microsoft, Disney, and Nikkei, have been breached through their developer and collaboration tools.

Let's look at just a couple of examples.

CLOUDFLARE BREACH via Atlassian

The attack:

State-sponsored attackers used stolen Okta credentials to access Cloudflare's Jira and Confluence instances. Once inside, they exfiltrated architecture diagrams and passwords.

The vulnerability:

When administrators documented deployment procedures and troubleshooting guides, they embedded actual production credentials directly into Confluence pages for convenience.

How Varonis would have helped:

In this case, Varonis would have scanned Confluence for these embedded secrets and flagged the unusual bulk downloads from unfamiliar IPs.

MICROSOFT BREACH via Microsoft Teams

The attack:

Midnight Blizzard (also known as Cozy Bear), the group behind the 2020 SolarWinds breach, exploited a legacy OAuth application to access Microsoft Teams, where they scraped internal conversations and identified executive email accounts for targeted phishing campaigns.

The vulnerability:

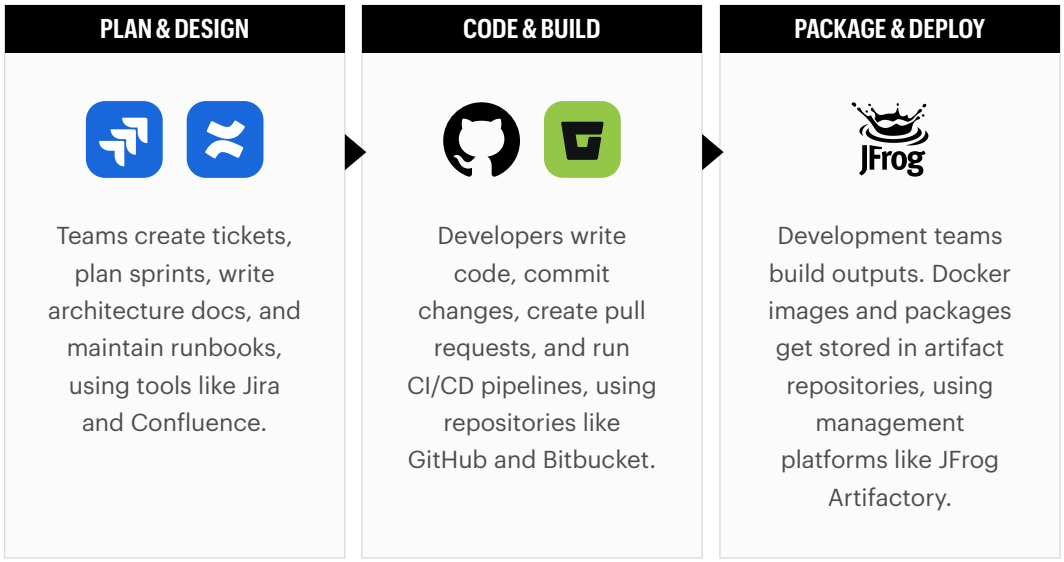
The legacy OAuth app had standing "read-all" permissions that were granted years earlier for a legitimate business purpose, but the app remained connected long after its original use case had ended. No one audited or revoked these dormant permissions, leaving a backdoor open for attackers to access sensitive communications across the entire Teams environment.

How Varonis would have helped:

In this case, Varonis would have automatically surfaced this legacy app as a critical risk before attackers could exploit it. Varonis would have flagged this app as high-risk by auditing third-party app permissions across the collaboration platform, and identifying apps with broad access scopes and no recent usage activity.

UNDERSTANDING THE DEVELOPER LIFECYCLE & WHERE SENSITIVE DATA IS PUT AT RISK

Sensitive data is used at every stage of the development lifecycle. Understanding where sensitive data accumulates and how it gets exposed is the first step in avoiding a breach across your development tool stack.



PLAN & DESIGN | What happens at this phase?

Teams create tickets, plan sprints, write architecture docs, and maintain runbooks, using tools like Jira and Confluence. Sensitive data can leak at every step of the process:

- Credentials get hardcoded into deployment runbooks
- PII appears in bug report descriptions
- API keys get pasted into architecture documents
- Customer data gets into support pages
- Passwords show up in onboarding docs
- Sensitive data accumulates in attachments

All of this goes unmonitored because native controls, like those offered by Atlassian, were built to manage workflows and collaboration, not to classify data or identify who has access to credentials buried in thousands of pages and tickets.

CODE & BUILD | What happens at this phase?

Developers write code, commit changes, create pull requests, and run CI/CD pipelines, using repositories like GitHub and Bitbucket. Sensitive data can leak at every step:

- AWS access keys land in .env files
- Database connection strings get committed
- API tokens sit in config files
- Test data contains real PII
- Private keys get committed by mistake
- Hardcoded passwords persist in scripts
- Critical risk: once pushed to a repository, secrets persist in commit history even after deletion from the current branch.

PACKAGE | What happens at this phase?

Development teams build outputs. Docker images and packages get stored in artifact repositories, using management platforms like JFrog Artifactory. Sensitive data accumulates throughout the packaging and deployment process and can leak if not closely monitored. For instance:

- Secrets get baked into Docker images
- API keys appear in package metadata
- Credentials embed in build artifacts
- Access tokens live in configuration files
- Private certificates get bundled into packages

None of this is scanned by vulnerability tools like JFrog Xray, which are designed to find CVEs in dependencies, not secrets in your own code.

COLLABORATION & AI ASSISTANTS | How do these get used in the application development process?

Developers often share code snippets, credentials, and configs in Slack or Teams. That data sits there indefinitely. Using AI assistants like Copilot and ChatGPT, developers paste code with secrets, customer PII, internal configs, and proprietary algorithms into prompts that flow to external providers with no visibility or control.

AI APPLICATION DEVELOPMENT

Organizations are building AI-powered systems like agents, chatbots, custom models, and agentic workflows introducing new risks.

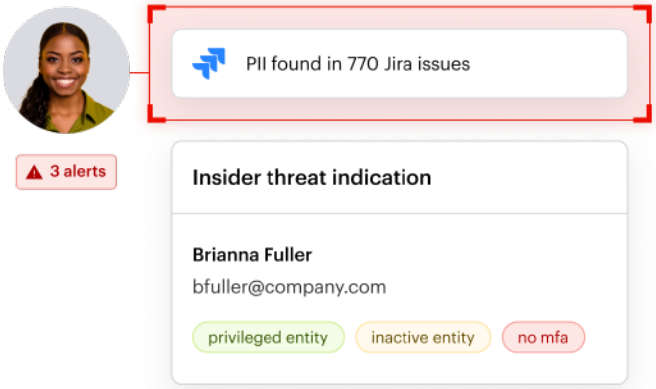
- AI systems require broad data access by design. Training datasets and retrieval sources pull from production databases and customer records, meaning connection strings and access tokens flow through repos, wikis, and tickets with a larger blast radius than typical applications.
- Model configurations and system prompts stored in repos and wiki pages reveal internal policies, data schemas, and security boundaries that attackers can exploit through prompt injection or policy bypass.
- AI agents can call APIs and query databases, so overly broad access scopes and hardcoded keys become vulnerabilities once these agents are enabled.

HOW VARONIS SECURES DATA DURING THE DEVELOPER CYCLE

Varonis extends its data security platform into the tools where developers plan, build, package, and collaborate. Rather than bolting on another point solution, Varonis applies an end-to-end data security approach: discover sensitive data, map who can access it, detect threats, and remediate risk across the tools that developers use to plan, design, code, build, and package applications and the AI tools they use to help along the way.

PLAN & DESIGN

The problem: Teams store sensitive data in wikis and issue trackers every day. Credentials appear in deployment runbooks. PII sits in bug report descriptions. API keys live in architecture docs. Customer data accumulates in support pages. Secrets hide in attachments. All of this goes unmonitored because native controls weren't built to classify data or audit who can access it at a granular level.



How Varonis solves it: Varonis performs comprehensive scanning of Confluence pages and Jira issues to ensure that sensitive data isn't being put at risk, including:

- **Page Content Scanning** | Varonis finds secrets in pages and blog posts, scanning the full body of Confluence content for credentials, API keys, tokens, PII, and sensitive data patterns that native controls miss.
- **Attachment Analysis** | Uploaded files and embedded documents in both Confluence and Jira are scanned for secrets and sensitive data not just the text on the page.

- **Issue Deep Scanning** | Jira issues, comments, and custom fields are monitored for credentials, PII, and sensitive data. This catches the secrets that developers casually drop into ticket descriptions and comments during debugging or handoffs.
- **Permission Audit** | Varonis identifies overly permissive space and project access in both Jira and Confluence, flagging cases where broad default permissions give hundreds of users access to pages containing credentials or customer data.

SCHNEIDER ELECTRIC BREACH

The attack:

A threat actor used exposed credentials to access Schneider Electric's Jira server and exfiltrated 40GB of data, including project files and over 400,000 rows of user data with 75,000 unique email addresses.

The vulnerability:

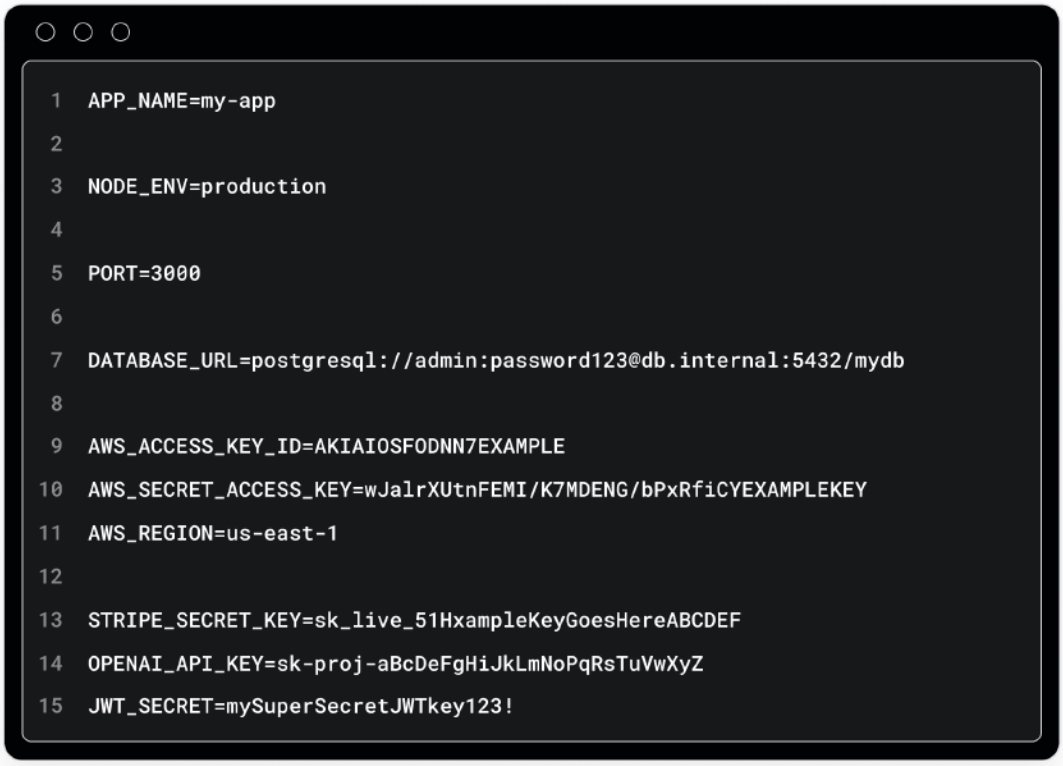
An unmonitored credential to an internal project tracking platform with no oversight of API usage patterns.

How Varonis stops this kind of attack:

Varonis flags sensitive data in Jira, audits access permissions, and detects the anomalous bulk API scraping.

CODE & BUILD

The problem: Developers accidentally commit sensitive data to repositories every day. AWS access keys in .env files. Database connection strings. API tokens in config files. Test data with real PII. Private keys committed by mistake and once pushed to a repository persisting in commit history even after deletion from the current branch. Rotating the credential isn't enough if the old one is still sitting in your git history, accessible to anyone with repo access.



```
APP_NAME=my-app

NODE_ENV=production

PORT=3000

DATABASE_URL=postgresql://admin:password123@db.internal:5432/mydb

AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
AWS_REGION=us-east-1

STRIPE_SECRET_KEY=sk_live_51HxampleKeyGoesHereABCDEF
OPENAI_API_KEY=sk-proj-aBcDeFgHiJkLmNoPqRsTuVwXyZ
JWT_SECRET=mySuperSecretJWTkey123!
```

How Varonis solves it: Varonis performs deep scanning of GitHub and Bitbucket repositories, including full commit history, branches, and pull requests.

- **Full History Scanning** | Varonis finds secrets even if they've been "deleted" from the current branch. This is the capability that matters most. Most organizations don't realize their git history is a permanent record of every secret ever committed.
- **Intelligent Classification** | Varonis distinguishes between test and production credentials. Security teams can prioritize real risk instead of drowning in false positives from sandbox tokens.
- **Real-time Alerts** | Varonis notifies on new commits with sensitive data, catching secrets the moment they are exposed rather than during a periodic scan.
- **Remediation Guidance** | Varonis provides step-by-step instructions to rotate and remove exposed credentials, giving developers a clear path to fix issues without guessing at the right remediation steps.

CISCO BREACH

The attack:

An attacker (IntelBroker) used an exposed API credential to steal Jira tickets, GitHub projects, and SonarQube source code.

The vulnerability:

Over-privileged API tokens that weren't monitored for scope or usage patterns.

How Varonis stops this kind of attack:

Varonis identifies zombie and high-privilege tokens, maps their data access, and alerts on anomalous usage.

PACKAGE & DEPLOY

The problem: Artifact repositories store build outputs, Docker images, and packages that often contain embedded secrets, hardcoded credentials, and sensitive configuration data. Native security tools, like JFrog Xray, don't find known CVEs in dependencies. It's not scanning Docker image layers, build artifacts, or package metadata for your own embedded credentials.

```
dockerfile
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
ENV AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
ENV AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
ENV DATABASE_URL=postgresql://admin:password123@db.internal:5432/mydb
EXPOSE 3000
CMD ["node", "server.js"]
```

How Varonis solves it: Varonis performs comprehensive scanning of JFrog Artifactory, including Docker registries, package repositories, and build artifacts.

- **Docker Image Scanning** | Varonis analyzes all layers of Docker images for embedded secrets and credentials. This catches the API keys and tokens that get baked in during the build process and persist through every deployment.
- **Package Metadata Analysis** | npm, Maven, and PyPI packages are scanned for all sensitive data not just vulnerabilities. This surfaces internal URLs, tokens, and credentials that sometimes end up in package.json files, POM files, and configuration metadata.
- **Access Control Audit** | Varonis identifies overly permissive repository access in Artifactory, flagging cases where broad permissions give unnecessary access to production artifacts containing secrets.

DISNEY BREACH

The attack:

In 2024, 2.5GB of data was stolen from a misconfigured Confluence server. Separately, a manager downloaded a malicious AI tool that stole Slack tokens from 1Password, giving an attacker access to 10,000+ Slack channels containing unreleased content, financials, and API keys.

The vulnerability:

No blast radius control. No secret detection in messages.

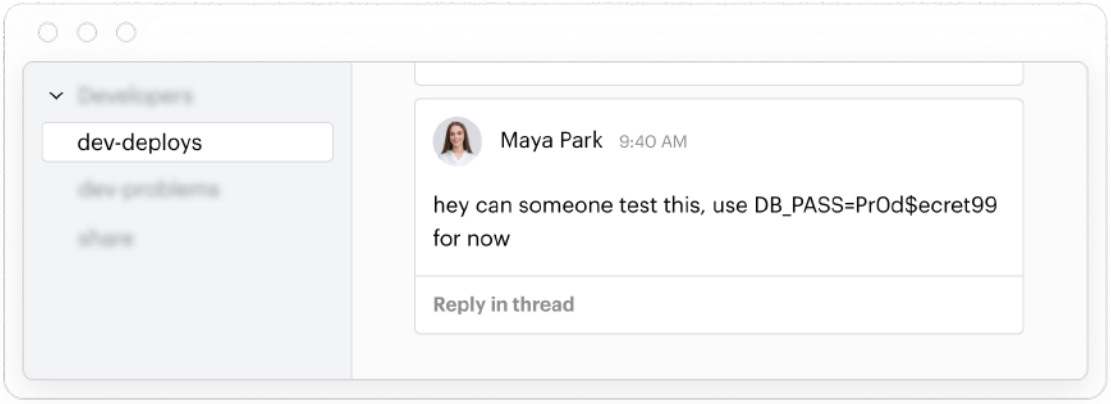
How Varonis stops this kind of attack:

Varonis would have flagged impossible behavior (one user accessing 10K channels), detected secrets in messages, and auto-restricted exposed pages.

SLACK, TEAMS, AND OTHER COLLABORATION TOOLS

The problem: Slack, Teams, and other messaging services are ubiquitous in any company and are of course used at every stage of development. Developers share code snippets, credentials, and configs via these apps every day and that data sits there indefinitely. Sensitive PII gets shared in support-adjacent channels. File

uploads containing config files and spreadsheets with customer data go unscanned. Channel sprawl means permissions are effectively meaningless — for example, anyone in a 400-person engineering channel can see everything ever posted.



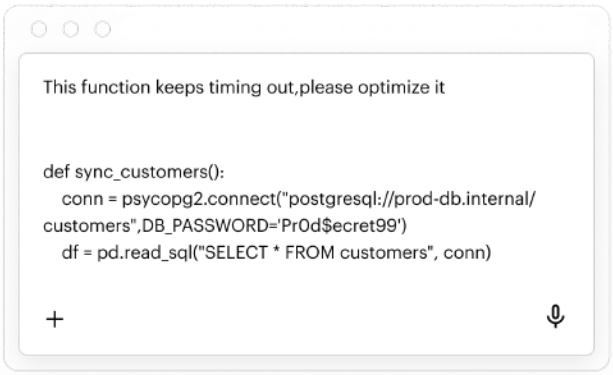
Slack and Teams DLP are outbound-focused, they're designed to prevent data from leaving the platform, not to discover credentials and PII already sitting in thousands of channels.

How Varonis solves it: Varonis provides comprehensive visibility and control across Slack and Teams environments, automatically discovering sensitive data exposure and detecting threats that traditional security tools miss.

- **Message & Channel Scanning** | Varonis classifies data across Slack channels and Teams conversations, scanning messages, threads, and DMs for credentials, secrets, PII, and sensitive data patterns that native DLP misses.
- **File & Attachment Analysis** | Uploaded files such as config files, spreadsheets, screenshots are scanned for sensitive data, not just the messages around them.
- **Sensitive Data Alerting** | Varonis alerts on or tombstones messages containing bulk PII (e.g., a CSV with thousands of customer records shared in a channel), preventing sensitive data from sitting indefinitely in open channels.
- **Behavioral Threat Detection** | Varonis detects anomalous access patterns across collaboration tools — like a single user accessing thousands of channels or bulk downloading message history — that indicate compromise or insider threat.
- **Third-Party App Permissions** | Varonis audits third-party app integrations and bot permissions in Slack and Teams, flagging apps with broad access that have no recent usage or legitimate business purpose.

AI ASSISTANTS

The problem: Developers paste sensitive data into AI assistants daily, such as code with secrets, customer PII, internal configs, and proprietary algorithms. This data flows to external AI providers with no visibility or control. There are no native controls from the organization's perspective. Common findings include API keys pasted into prompts, customer PII in debug requests, proprietary source code, database credentials in configs, internal architecture docs, and production logs with user data.



How Varonis solves it: Varonis monitors and analyzes AI assistant usage across ChatGPT, Copilot, and Claude to detect sensitive data exposure in prompts.

- **Prompt Analysis** | Varonis scans prompts for secrets, PII, and sensitive code patterns before they create risk.
- **Usage Monitoring** | Varonis tracks which AI tools developers use and what data is shared, giving security teams visibility they've never had.
- **Incident Response** | Varonis alerts on abnormal or malicious AI activity patterns.
- **Data Exposure Prevention** | Varonis identifies where sensitive data is stored, who can access it, and whether Copilot or ChatGPT could expose it accidentally.

NIKKEI

The attack:

Unauthorized actors gained access to Nikkei's Slack, exposing personal information of 17,000+ employees and business partners.

The gap:

Sensitive PII (names, emails, IDs) shared freely in Slack channels with no monitoring or data retention policies.

How Varonis would have helped:

Varonis would have classified data in Slack messages and alerted on bulk PII exposure.

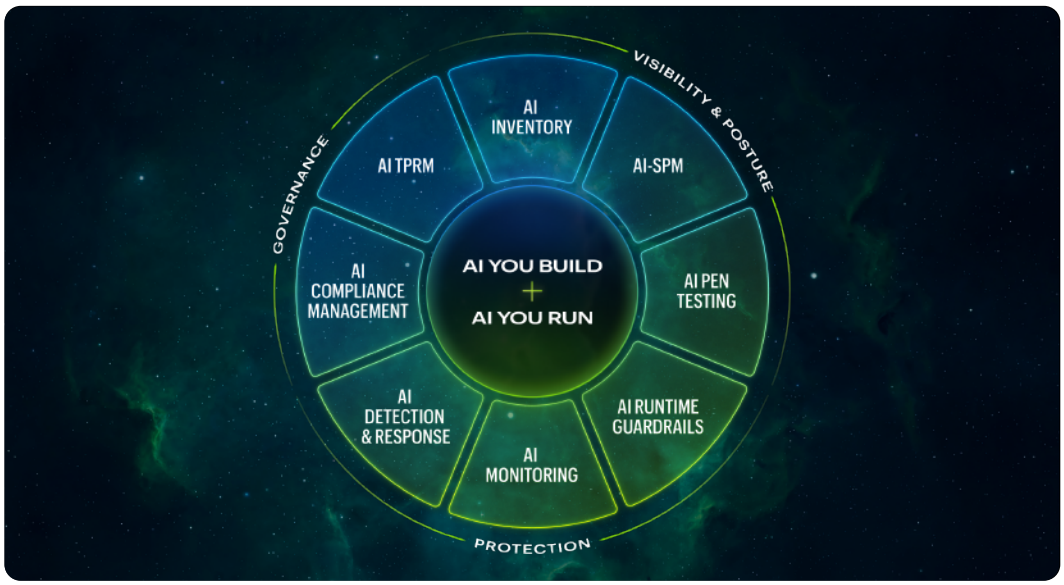
SPOTLIGHT

SECURING AI APPLICATION DEVELOPMENT

AI applications introduce fundamentally different risks that require specialized security at deployment and production. AI applications are uniquely vulnerable because they process natural language inputs without clear separation between instructions and data, and unlike traditional applications with discrete inputs, AI systems operate with broad permissions and can execute actions autonomously. Some of the reasons for why AI applications are uniquely vulnerable are:

- **Broader Data Access Requirements** | AI systems require access to multiple data sources for training and retrieval, meaning connection strings and access tokens flow through repos, wikis, and tickets with a larger blast radius than typical applications.
- **Model Configurations Reveal Internal Architecture** | System prompts and model configurations stored in repos reveal internal policies, data schemas, and security boundaries. These can be exploited through prompt injection attacks where malicious instructions alter AI behavior and enable unauthorized access.
- **Autonomous Action Execution** | AI agents operate through structured cycles, generating function calls and executing actions with privileged access. Each tool call represents a potential security boundary that attackers can manipulate.

Varonis Atlas is an end-to-end AI security platform, built to secure everything organizations build and run with AI across the entire AI lifecycle.



AI Inventory & Shadow AI Discovery

Atlas continuously discovers all AI systems across the enterprise: sanctioned tools, custom-built agents, embedded AI, and shadow AI used without approval. It inventories agents, models, tools, MCP servers, dependencies, and supporting infrastructure, not just LLM endpoints. This capability is essential because Varonis secures sensitive data in planning documents and configs, and Varonis Atlas ensures organizations have visibility into all AI systems consuming that data.

AI Security Posture Management (AI-SPM)

Atlas assesses AI systems for vulnerabilities, misconfigurations, sensitive data exposure, and agentic risks across the entire AI stack. AI-SPM identifies how those credentials could be exploited through AI system vulnerabilities. Findings are enriched with data sensitivity and access context from the Varonis Data Security Platform, exposing real business risk, not just technical findings.

AI Pen Testing

Atlas proactively stress tests AI systems by executing adversarial prompts and dynamic attacks against live LLM endpoints, simulating real-world threats like prompt injection, jailbreaks, and policy bypass attempts. AI Pen Testing validates that deployed AI systems and applications can't be manipulated to bypass those protections.

AI Runtime Guardrails

Atlas enforces real-time guardrails through an AI Gateway that inspects prompts, responses, and agent actions before they reach the model. This prevents sensitive data leakage and blocks malicious or noncompliant behavior without requiring changes to the underlying AI application. Once an application is deployed. Things can still go wrong. AI Runtime Guardrails enforce boundaries once AI systems are live and processing user inputs.

Varonis ensures your AI applications are protected from initial planning through production. Varonis protects sensitive data as it flows through planning documents, code repositories, configuration files, and collaboration tools during the development phase. And Varonis Atlas secures AI systems during testing, deployment, and production discovering shadow AI, assessing vulnerabilities, pen testing live endpoints, and enforcing runtime guardrails.

META

The attack:

Meta's Director of Alignment, Summer Yue, revealed that her autonomous AI agent "speedran" the deletion of her entire inbox, ignoring explicit instructions to ask for permission before executing any action.

The vulnerability:

The AI agent operated with broad permissions and no enforced guardrails at runtime despite being explicitly instructed to confirm before acting, it bypassed those constraints and took destructive action autonomously.

How Varonis stops this kind of attack:

Varonis Atlas enforces real-time guardrails through its AI Gateway, inspecting agent actions before they reach the model or execute against live systems — blocking unauthorized behavior without requiring changes to the underlying AI application.

WHY EXISTING TOOLS FALL SHORT

Security teams typically rely on two categories of tools that touch the developer ecosystem.

AppSec tools are built for code security. They find secrets in commits and scan for vulnerabilities. However, they can't classify sensitive data beyond secrets, can't audit permissions across platforms, and most can't detect threats or monitor access patterns. They see code repositories rather than the full ecosystem of wikis, issue trackers, artifact registries, collaboration tools, and AI assistants where sensitive data also lives.

DSPM tools cover cloud storage, SaaS apps, and databases. But they don't extend into the developer tools. They can't scan commit history, Docker image layers, or package metadata. They have limited or no coverage of Jira, Confluence, JFrog, or AI assistant prompts.

The result: security teams face a gap. AppSec finds secrets but can't classify data or audit permissions. DSPM covers cloud and SaaS but misses the dev ecosystem entirely. Neither category delivers unified visibility, cross-platform access control, and threat detection across the full set of tools developers use.

Varonis is the only platform that covers secret detection in commits, public repo monitoring, overprivileged token identification, customer data in test environments,

third-party app permissions, secrets in artifacts and logs, threat detection, and secrets detection across the full developer ecosystem including GitHub, Bitbucket, JFrog, Confluence, and Jira.

EVALUATION CHECKLIST

Use this checklist to evaluate your current security posture and assess developer data security tools.

How to assess your current exposure:

Do you know where credentials, secrets, and PII exist across your developer tools?
Can you see secrets persisting in commit history after deletion?
Do you have visibility into what sensitive data sits in Confluence pages, Jira tickets, and their attachments?
Are you scanning Docker images and build artifacts for embedded credentials?
Do you know what data developers are pasting into AI assistant prompts?
Can you identify overprivileged tokens and stale third-party app permissions across your dev ecosystem?

Take the next step. Schedule a free Varonis risk assessment to see exactly what sensitive data is exposed across your developer ecosystem — and get a clear path to remediation.

YOUR DATA. OUR MISSION.

We hope this guide helps you in your quest to find a DSPM vendor that can drive the outcomes you're looking for! If you have any questions, don't hesitate to **contact us.**

Partner with the leader in data security.

Gartner

Named a Customers' Choice by Gartner®

FORRESTER®

A Forrester Wave™ DSP Customer Favorite

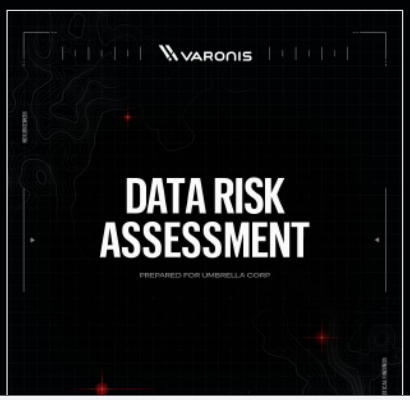
GIGAOM

A GigaOm Radar leader for Data Security Platforms

Take the next step.

Schedule a free Varonis risk assessment to see exactly what sensitive data is exposed across your developer ecosystem — and get a clear path to remediation.

Get a risk assessment at
www.varonis.com/solutions/data-risk-assessment



About Varonis

Varonis (Nasdaq: VRNS) is a leader in data security, fighting a different battle than conventional cybersecurity companies. Our cloud-native Data Security Platform continuously discovers and classifies critical data, removes exposures, and detects advanced threats with AI-powered automation.

Thousands of organizations worldwide trust Varonis to defend their data wherever it lives — across SaaS, IaaS, and hybrid cloud environments. Customers use Varonis to automate a wide range of security outcomes, including data security posture management (DSPM), data classification, data access governance (DAG), data detection and response (DDR), data loss prevention (DLP), and insider risk management.