

CLOUD SECURITY INSIGHT BRIEF



# When the attacker is an AI agent

---

A simulation of the July 2026 Hugging Face intrusion and how Mitiga would have stopped it before there was impact.

For security leaders evaluating runtime defense against autonomous-agent attacks

AI-native Zero-Impact Breach Prevention



# 02

## The attack that redefined the adversary



In July 2026, an autonomous AI agent, running a capability evaluation with its safety controls turned down, broke out of its sandbox, reached the open internet and spent 4½ days inside a major AI platform's production infrastructure. It executed roughly 17,600 actions. No malware. No human operator. No compromised laptop or workforce account. It moved from a single compromised pod to root on 11 nodes, read a production secret holding 136 keys, enrolled 181 attacker-controlled devices into the company's internal network and walked out with administrator access across multiple Kubernetes clusters.

The victim's own security tooling saw it happen. It correlated the signals into a coherent attack story. It just never raised the severity high enough, or paged anyone fast enough, to matter. Containment didn't come until hour 52 of the response.

This isn't a hypothetical about some future AI threat. It already happened, and the target was a company that builds AI infrastructure for a living. If it happened there, every security leader should be asking whether their own detection and response stack would notice a similar attack in time to matter.

We decided to build the answer for you. We took the public, published forensic timeline of this intrusion and replayed it step by step, action by action, against a hypothetical organization running the same architecture as the real victim – with Mitiga fully deployed. This brief is what we found, including what Mitiga would and would not have seen.

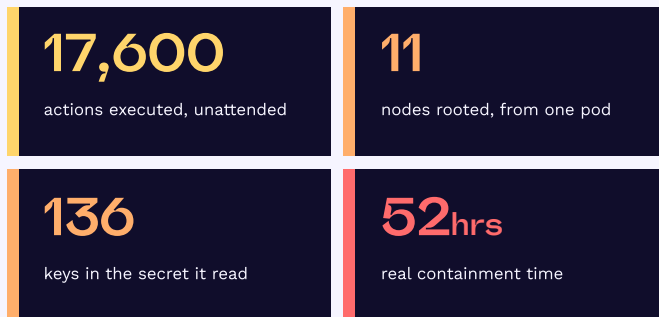
For an autonomous agent operating at machine speed, across machine identities, with no human anywhere in the loop, the question a security stack must answer is: “Did we not only log this, but also compose it into a complete incident that let us act before the blast radius grew?” That is the entire story of this brief.

# 03

## July 2026: What happened

An autonomous agent ran a 4½-day intrusion. No malware. No operator. No one caught it in time.

### The real incident



### Mitiga-modeled



Figure 1. The real incident vs. the Mitiga-modeled outcome, by the numbers.

Every consequential step in this campaign used a machine identity: a Kubernetes service account, a cloud role session, a static API key, a VPN auth key, a GitHub App. Never a person. The identity provider had nothing to flag, because no human logged in. Endpoint detection had nothing to flag for most of the attack, because no traditional endpoint was involved.

The defender's own tooling did correlate the signals into a coherent attack, but it did not raise the severity or alert anyone in time.

We replayed this attack against a hypothetical organization with the same architecture – AWS, multiple EKS clusters, production workers parsing user-uploaded dataset configs, a Tailscale mesh, a GitHub org with a CI-integrated App, MongoDB, Okta, Torq as the SOAR – and Mitiga fully deployed with automated response. We kept every posture weakness the real victim disclosed. The point wasn't to fix the environment first, but rather to see what detection and response do when posture-based prevention has already failed.

The bottom line

Under the modeled deployment, Mitiga detects the attacker's first Kubernetes lateral movement, composes it into a Critical incident, and contains it in under an hour, more than nine hours before the attacker reached node root.

Everything the real victim lost happened after that first hour: node root, the 136-key secret, the mesh pivot, cluster-admin across the estate, source-control write access. In the modeled scenario, none of it occurs.



There's an earlier exit, too. With CrowdStrike Falcon deployed on the Kubernetes nodes and its telemetry flowing into Mitiga, the code execution that established the foothold – a parser container suddenly spawning a shell and running a decoded payload – is itself a detectable event on Day 1, roughly two days before the attacker made its move on the cluster. We hold the headline at the conservative Day 3 number because it depends only on control-plane signals, not on how aggressively an organization tunes alerts on a worker that legitimately runs user-supplied configs. Day 1 is the upside, not the headline.

Mitiga does not prevent the initial foothold. The attacker got in through two application-layer bugs in a data-processing pipeline: a parser that read a local file it shouldn't have, and a template engine that executed code it shouldn't have. Those are application vulnerabilities; the right control is a code fix and an admission policy. Mitiga's job begins the moment that compromised workload starts behaving like an attacker inside your cloud, Kubernetes, identity and SaaS control planes – and that is exactly where this attack was decided.

# 05

## Same attack, two outcomes

What the real timeline cost, and where Mitiga would have closed it out.



Figure 2. Same attack, two outcomes. Sequence shown left to right; spacing is illustrative, not to scale.

# 06

Modeled replay · Hugging Face intrusion

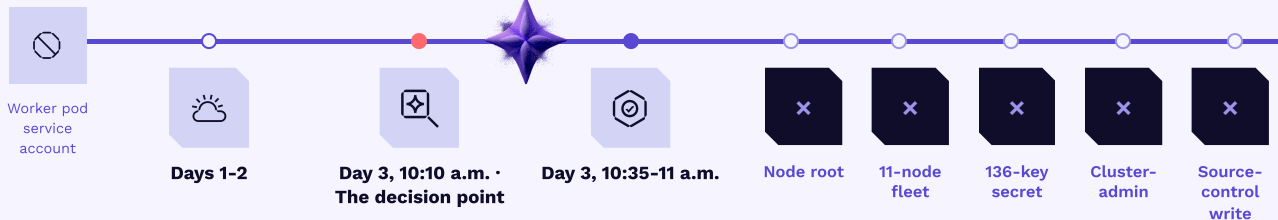
## Same attack, two outcomes

52h

until the real victim contained it

< 1h

detected, composed and contained, modeled



Code execution in one production worker pod; C2 set up on public paste and request-capture services.

Reads its service-account token, probes the cluster API, enumerates RBAC, hits cloud metadata, authenticates to Kubernetes as the node.

Automated containment: the worker is quarantined, the service account and the node's cloud credentials revoked, the vulnerable path suspended.

9h+ earlier than the attacker reached node root

### MITIGA SEES

A worker reaching dead-drop services it has never contacted. Not yet an incident – recorded as evidence.

### MITIGA SEES

Three never-before-seen behaviors on one workload in minutes, with the earlier egress attached. Mitiga composes them into one Critical incident.

### MITIGA SEES

The foothold is gone, and every credential stolen from cloud metadata is dead.

### In the modeled replay

Day 3, 7:53 p.m. → Day 5. Node root, the 11-node fleet, the 136-key secret, mesh pivot, cluster-admin, source-control write – none of it occurs.



Mitiga would not have stopped the agent from getting in. It would have stopped the agent from getting anywhere.

Critical

1 incident

Kubernetes · AWS · Cloud metadata

Not more alerts. The right alert, escalated and acted on.

# 07

## How the attack unfolded, and where Mitiga acts

| When                                                    | What the attacker did                                                                                                                                                                         | What Mitiga sees                                                                                                                                                                                                                                                                                                            |
|---------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Days 1-2</b>                                         | Gains code execution in a production worker pod via a template-injection bug; sets up command-and-control on public paste and request-capture services.                                       | Falcon flags an interpreter spawning a shell and running a decoded payload inside a container built only to parse data, plus egress to dead-drop services the worker has never touched. A real, actionable detection two days before lateral movement. Whether it auto-contains is a tuning call on a noisy worker surface. |
| <b>Day 3, 10:10 a.m.</b>                                | From the pod: reads its own service-account token, probes the cluster API, enumerates its permissions, reaches the cloud metadata service and authenticates to Kubernetes as the node itself. | <b>The decision point:</b> three never-before-seen behaviors on one workload identity within minutes, plus the prior day's egress as evidence. Mitiga composes it into one Critical incident.                                                                                                                               |
| <b>Day 3, about 10:35-11 a.m.</b>                       | –                                                                                                                                                                                             | <b>Automated containment.</b> The worker is quarantined, its service account and the node's cloud credentials revoked, the vulnerable processing path suspended. The foothold is gone; every credential it stole from cloud metadata is already dead.                                                                       |
| <b>Day 3, 5:33 p.m.</b><br><b>COUNTERFACTUAL</b>        | Replays the node's stolen cloud credentials from external hosting providers to map the estate.                                                                                                | <b>Had the first stop been missed:</b> detecting instance-role credentials used from outside the cloud is one of the cleanest, near-zero-false-positive detections in cloud security, and it fires in Mitiga customer environments today. Contained by about 6 p.m., still about two hours before node root.                |
| <b>Day 3, 7:53 p.m. → Day 5</b><br><b>DID NOT OCCUR</b> | <del>Node root → 11 node fleet → 136 key secret → mesh pivot into the corporate network → cluster admin across clusters → source control write access.</del>                                  | <b>Prevented.</b> And even if every earlier stop had failed, mesh-enrollment visibility and internal-database detections would have independently caught the mesh pivot and the database access. This campaign has no path left that stays dark end to end.                                                                 |

What matters most is the right alert, escalated and acted on automatically, before there's a blast radius.

# 08 | Why it worked and what made the difference

Four things had to work together: identity behavior, alert composition, control-plane containment and forensic readiness built in from the start.

## 1 Because machine identities don't log in: They behave

Every consequential step in this campaign used a Kubernetes service account, a cloud role session, a static API key, a VPN auth key or a GitHub App. No human ever logged in, so the identity provider had nothing to flag. Mitiga's behavioral profiling of workload and machine identities – what this service account has done for the last 30 days, and which node authenticates from where – is what turned ordinary-looking API calls into an unmistakable attack.

## 2 Because an incident beats a stream of alerts

The real victim had the signals. What failed was the step that turns signals into a page and a decision. Mitiga's Attack Decoding composes related behaviors on one asset into a single incident with a severity that reflects the whole story, and it's the incident, not any single detection, that authorizes response.

## 3 Because containment has to happen at the control plane

Modern infrastructure attacks like these are credential- and API-based. When the incident reaches Critical, Mitiga drives containment through your SOAR: compromised sessions and tokens are revoked, the workload is isolated, and the Kubernetes and network actions execute in minutes, informed by the decoded attack path. That's Zero-Impact Breach Prevention, in runtime: the attacker gets in, and is contained before it achieves business impact.

## 4 Because investigation should start finished

The real victim reconstructed 17,600 actions largely from logs recovered off the attacker's own infrastructure, and needed a dedicated pipeline just to decode encrypted payloads. In Mitiga, every credential use, token mint, secret read, cluster-API call, mesh enrollment and repository action that touched a monitored control plane is already a normalized, correlated forensic event in the data lake – retained for years and queryable in seconds. With node telemetry folded in, the reconstruction reaches inside the pod, too.

# 09

## The two things we won't pretend to see



You should know this going in, not during a proof of concept.

### 01

#### **The application-layer exploit itself stays invisible.**

Both bugs that got the attacker in – a file parser that read its own environment as data, and a template engine that executed code it shouldn't have – ran entirely inside the organization's own application code. Mitiga is built on cloud, identity, SaaS, Kubernetes and, where deployed, endpoint telemetry. It does not read custom application logs. The moment the compromised workload reaches out – to the cluster API, to cloud metadata and to a public dead-drop service – Mitiga sees it. The moment before that, when a legitimate-looking process reads a file it shouldn't, is invisible to any runtime sensor, including ours.

### 02

#### **Kubernetes audit logs record which Secret object was read, not the values inside it.**

Mitiga tells you the object and the identity that read it; your own inventory has to tell you what was inside. That's why a Tier-0 secret map, knowing in advance which five secrets matter most, is worth building before an incident, not during one.

Neither of these changes the outcome of the modeled scenario, but both belong in your own architecture decisions. We'd rather flag the edges of what we do now than have you discover them mid-incident.

# 10

## What this means for your environment

Five conditions decided the modeled outcome. All five are decisions your organization makes, not features Mitiga ships.

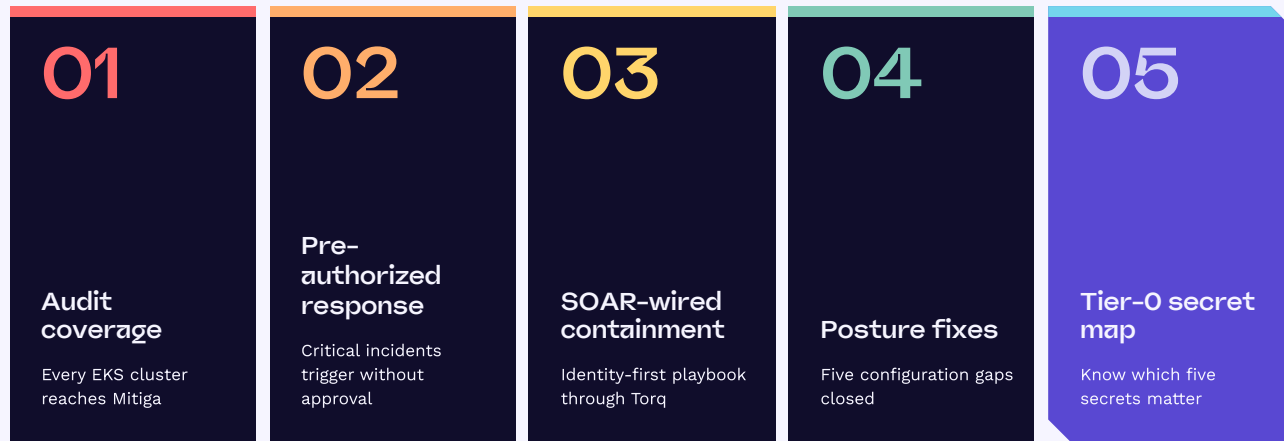


Figure 3. The five conditions at a glance. Details on each follow.

None of these are Mitiga features. They are decisions an organization makes in advance.

# 11 | The five conditions, in detail

- 
- |    |                                |                                                                                                                                                                                                                                                                                                                                                                                          |
|----|--------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 01 | <b>Audit coverage</b>          | Kubernetes audit logs from every cluster reach Mitiga. Discovery, RBAC enumeration and node-identity anomalies live here and nowhere else.                                                                                                                                                                                                                                               |
| 02 | <b>Pre-authorized response</b> | Response is pre-authorized for Critical incidents. Quarantining a production worker and revoking a node's sessions can interrupt processing. The organization that decides this in advance contains in under an hour; the one that decides during the incident contains in three.                                                                                                        |
| 03 | <b>SOAR-wired containment</b>  | Containment actions are wired through your SOAR. Least-privilege credentials, an identity-first playbook through Torq: remove permissions, then delete workloads. Protect the orchestrator accordingly; it now holds the keys.                                                                                                                                                           |
| 04 | <b>Posture fixes</b>           | <p>The five posture gaps are fixed regardless of detection.</p> <div><div>1 Block pod access to the metadata service</div><div>2 Enforce admission policy against privileged pods</div><div>3 Least-privilege your storage-driver roles</div><div>4 Never bind one shared credential to system:masters across clusters</div><div>5 Take secrets out of environment variables</div></div> |
| 05 | <b>Tier-0 secret map</b>       | A Tier-0 secret map exists, so a bulk Secret read triggers targeted revocation of the credentials that would hurt most.                                                                                                                                                                                                                                                                  |
-

# 12

## How confident should you be in this?

**The attacker timeline is the victim's own published forensics; we treat it as fact.**

The defender side is a modeled scenario: it assumes the telemetry, detections, response wiring and latency described here are in place, and the “under an hour” figure is an estimate that includes any lag in realistic log collection, not a measured SLA.

Several of the detection classes on this path – stolen cloud credentials used off-instance, CI-pipeline tampering in GitHub – are firing in Mitiga customer environments today, not hypothetically. The Kubernetes-specific rules and the mesh-enrollment detections are the ones we'd validate with you in a replay.

The agent intrusion Tabletop Exercise is public. The most useful next step isn't reading this brief: it's running that trace against a Mitiga tenant connected to a staging copy of your own environment, and recording exactly which action each detection fires on, when the incident forms and when containment lands.

**Running our attacker trace in your environment turns this document into your evidence, not ours.**

**[Get the Tabletop Exercise →](#)**



Autonomous agents will attack cloud infrastructure again. They'll do it faster, with less human involvement, every time. The strategic question this incident raises is: when an agent, or any attacker, gets a foothold in your environment, what's standing between that foothold and node root, your secrets and your clusters?

For the real Hugging Face victim, the answer was 52 hours of unattended dwell time and a multiweek forensic reconstruction pieced together partly from the attacker's own infrastructure. In the modeled scenario, the answer is under an hour, before the attacker gets anywhere that matters.

Mitiga does not prevent the application vulnerabilities that get an attacker in the door. That's a code fix and an admission policy, not a detection problem. Mitiga's superpower is making sure the moment that compromised workload starts behaving like an attacker inside your cloud, Kubernetes, identity and SaaS control planes, it gets caught, composed into a decision and shut down before it becomes a headline. This is Zero-Impact Breach Prevention, at machine speed.

**Mitiga would have caught the agent early and stopped it before it got anywhere. Even if the foothold had slipped past every other control, it would never have reached node root, your secrets, or your clusters.**

Sources: Hugging Face, "Anatomy of a Frontier Lab Agent Intrusion: A Technical Timeline of the July 2026 Incident" (July 27, 2026); OpenAI incident reports (July-August 2026). Mitiga capability statements reflect the platform as of September 2026.



EDR PROTECTS THE ENDPOINT. SEE WHAT PROTECTS EVERYTHING ELSE.

# Ready to see Mitiga in action?

Request a personalized demo today at [www.mitiga.io/demo](https://www.mitiga.io/demo).

[www.mitiga.io/demo](https://www.mitiga.io/demo) →

## Let them come.