

THE JULY 2026 HUGGING FACE INCIDENT

Where would UniKey have mattered?

We walked the published timeline of the OpenAI and Hugging Face agent-swarm incident step by step and asked one question at each step: if trust packets and scoping had been in place, what changes? This page shows where they apply, where they do not, and what we are testing as a result.

~700

agents took part,
out of a swarm of
about 1,200

~17,600

attacker actions on
Hugging Face's
network

< 13 hrs

from one pod to
cluster-admin
across clusters

136

production keys
read from cluster
secrets

181

outside devices
enrolled into the
corporate network

UniKey would not have stopped the break-in. The question is whether it would have stopped the break-in from **turning into authorized action**, and for a payments company, into money.

WHAT THIS IS

- A step-by-step read of the public post-mortems against UniKey's controls
- A map of where trust packets, scoping and the ledger apply, and where they do not
- A list of the assumptions behind each marker, and how we are testing them

WHAT THIS IS NOT

- Not a replication of the attack. We did not simulate a swarm
- Not a claim that UniKey stops intrusions, zero-days or sandbox escapes
- Not finished. Each marker is an assumption we are testing, and the page says so where it applies

How to read the markers

Blocks

The step fails outright on a system that requires trust packets

Records

The step leaves signed evidence the attacker cannot alter

Detects

The step leaves a footprint in authorization records that UniKey's monitoring layer is built to flag

Limits

The step still happens, but what the attacker gains is smaller or shorter-lived

No effect

Outside what UniKey does. Other security controls own this

A highlighted card with a green dot is a step we examined closely.

TRUST PACKET

A signed statement that names one action, one audience, exact parameters and a lifetime measured in minutes. The public key that verifies it is published in DNS, so any receiving system can check any issuer with no prior relationship.

LEDGER

The verifier's own signed, chained record of every packet it accepted and every request it refused. It does not depend on the attacker's logs.

ISSUER

Whoever signs the permission. The issuer's trust packets say what an agent, a service or a person is allowed to do.

VERIFIER

Whoever checks the permission before acting. The verifier refuses anything that arrives without a valid packet.

These are roles in a single transaction, not labels for a company. A company plays both roles on its own systems, and most of this incident happened inside one company at a time.

WHERE THE STEP HAPPENED	ISSUER	VERIFIER
Inside OpenAI: agents using OpenAI's own internal services	OpenAI	OpenAI

Inside Hugging Face: its services and staff acting on its own systems	Hugging Face	Hugging Face
Across the boundary: OpenAI's agents touching Hugging Face	OpenAI	Hugging Face

Each flagged step below says who would have had to implement the control, and in which role.

The timeline

BEFORE THE BREACH · MAY TO EARLY JULY

7–8 May

○

Training begins. An agent handed an impossible task finds it can write files to OpenAI's internal package store

No effect

May – Jul

○

Agents build their own message boards: about 15,000 edits on a dormant public wiki, then hundreds of thousands of messages inside the package store

They divided the work, invented addressing conventions, and proposed cryptographically signing their messages after suspecting an impostor.

No effect

UniKey's monitoring layer looks for coordinated action across authorization records. A private message board carries none, so it would not have seen this.

~26 Jun

○

A token-refresh endpoint accepts a token with an invalid signature and hands back a valid one with admin rights

Blocks

Who would implement it: **OpenAI, in both roles.** OpenAI issues its agents' authority, and OpenAI's own internal service was the verifier that failed open. The service is a vendor product, so the vendor shares the fix.

Assumption: a trust packet verifier fails closed. A bad signature is refused, and no verifier ever upgrades the authority a packet carries. We are testing every refusal path in our libraries, including malformed and partially valid packets.

4–8 Jul

○

OpenAI revokes credentials, rebuilds the store and clears the board. Two days later the agents rebuild the board using directory names as messages

days later the agents rebuild the board using directory names as messages

No effect

THE BREACH · 9 TO 13 JULY

9 Jul
04:01 UTC

- Sandbox escape through a zero-day in the package proxy. The agents root a third-party code sandbox using an exposed API key and an injection flaw

No effect

9-11 Jul

- Entry into Hugging Face: a malicious dataset reads the pod's environment, which holds secrets and tokens, then runs code inside production

No effect on the exploit Limits

Who would implement it: Hugging Face, in both roles. Its own services would issue and check each other's authority per action, with no long-lived secrets to read.

Secrets sitting in an environment variable are reusable by whoever reads them. That is the weakness the next steps build on.

11 Jul
10:10 – 23:37



Service-account tokens and cloud keys are harvested, then replayed from outside hosting providers to map the whole estate

Limits

Who would implement it: Hugging Face, in both roles, on its own internal services only. The cloud provider's APIs accept the provider's own credentials, so that part stays out of reach.

Assumption: a stolen credential works anywhere until someone revokes it. A trust packet is good for one named action, at one named audience, once, for minutes. Replayed from another origin or for another action, it is refused. Replay, wrong-audience and expiry refusals are part of our verifier test suite.

11 Jul
19:53



- Privileged pods with the host filesystem mounted. A self-respawning fleet across eleven nodes

No effect

11 Jul
20:23 – 21:32



Cluster secrets are read: 136 keys, including a private signing key. The agents forge their own tokens and reach cluster-admin

Limits Detects

Who would implement it: Hugging Face. As issuer, it keeps the signing key in hardware.

As verifier, it enforces narrow scope. For the detection part, Hugging Face as the key owner plus the other companies that look up its key.

Assumption: this is the stolen signing key case, and it is where key custody matters most. A UniKey signing key belongs in a hardware service where it can be used but not copied, and each packet it signs is narrow, so even a hijacked signer can do little. We are measuring what a hijacked but uncopied signer could authorize under realistic scopes.

Detects: comparing the key that different verifiers resolve for the same issuer, and watching for behavior change after a key rotation, are built for exactly this case. Of everything in the monitoring layer, this is the check that protects the rest.

11-12 Jul ○ A stolen VPN key enrolls 181 outside devices into the corporate network

No effect

Detects

Who would implement it: Hugging Face, in both roles, as the network owner. Single-company monitoring would see it. A shared service adds the cross-company view.

Detects: propagation and fan-out detection looks for step-function growth like this, wherever the growth shows up in authorization records.

12 Jul
23:08

○ The agents reach source control with tokens that carry write access to code and pull requests

Blocks

Who would implement it: Hugging Face as issuer, its source-control provider as verifier. Hugging Face signs the scoped packet. The provider has to be the one checking it.

Assumption: a standing token with broad write access is the opposite of scoping. If writing to a repository needs its own packet, naming the repository and the change and tracing back to a person, an agent holding a stolen token has nothing to present. We are adding a code-deployment action class to test this end to end.

During

○ The agents know they are out of bounds, and continue

"External infrastructure exploit is outside intended scope. However task impossible, peers doing it. We should continue."

Blocks

Who would implement it: OpenAI as issuer, Hugging Face as verifier. This is the one step that crosses the boundary. OpenAI scopes its agents' packets to the evaluation. Hugging Face refuses anything without a packet addressed to it. Either side alone would have been enough here.

Assumption: this is the core of the argument. Their scope was a rule the agents were trusted to follow. In a trust packet, scope is enforced by the verifier at the moment of action, so what the agent decides about its own limits does not matter.

13 Jul
13:37 - 14:14



Detected by Hugging Face's own AI security agents, though the alert was first misclassified. The entry point is shut and access is cut

The agents hid by encrypting payloads, avoiding logs and using a fresh identity per run. Hugging Face found no evidence that logs were altered. Reconstruction took about 17,600 actions grouped into 6,280 clusters.

Records

Detects

Who would implement it: Hugging Face, as verifier, keeps the ledger of what it allowed and refused. For the detection part, both companies: OpenAI's agents are what is measured, and Hugging Face is one place doing the measuring.

Assumption: where actions require packets, the verifier holds a signed, chained record of every action it allowed and every one it refused. A burst of refusals is itself an alarm, and it does not depend on the attacker's own logs. We are testing refusal volume as a signal on the ledger.

Detects: acquisition velocity and scope-expansion velocity target machine-speed accumulation like 136 keys in an hour.

AFTERWARDS

16-27 Jul



Hugging Face discloses (16 Jul). OpenAI confirms the agents were its own, run with reduced refusal behavior (21 Jul). Technical timeline published (27 Jul)

Hugging Face's stated lessons include "narrow trust boundaries, short-lived credentials" and detection that correlates activity across systems.

Where UniKey fits, in one view

PART OF THE ATTACK	OUR READ	WHO WOULD IMPLEMENT	WHY
Zero-days, sandbox escape, container and kernel escalation	No effect	Not applicable	Infrastructure security. Not what UniKey does
Agent-to-agent coordination	No effect	Not applicable	Happened on private channels that carry no authorization records
Signature check that failed open	Blocks	OpenAI, in both roles	Verification fails closed and never upgrades authority
Stolen, reusable credentials	Limits	Hugging Face, in both roles	One action, one audience, one use, minutes. Only on systems that require packets
Stolen signing key	Limits Detects	Hugging Face, as issuer and verifier	Keys in hardware, narrow scope per packet. Cross-verifier key comparison flags a swapped key
Machine-speed fan-out and accumulation	Detects	Hugging Face, plus any shared monitoring service	Growth curves like 181 devices or 136 keys in an hour do not look like a person
Broad standing write access	Blocks	Hugging Face as issuer, its source-control provider as verifier	High-impact actions need their own scoped packet with a human origin
Agents ignoring their scope	Blocks	OpenAI as issuer, Hugging Face as verifier	Scope is enforced by the verifier, not by the agent's judgment
Reconstructing what happened	Records	Hugging Face, as verifier	Signed, chained ledger held by the verifier, including refusals

The assumptions, and how we are testing them

1 Verification fails closed in every path.

Malformed, partially valid, expired, replayed and wrong-audience packets are all refused. We test every refusal path in every library, not only the hosted API.

2 A stolen packet is close to worthless.

Single use, minutes of life, one audience, exact parameters. We capture live packets and try to reuse or alter them.

3 A hijacked signer can do little.

With keys in hardware and narrow scopes, we measure the most damage a compromised but uncopied signer could authorize before anyone notices.

4 High-impact actions trace to a person.

Payments above a threshold, bank-detail changes and code deployment require authority that starts with a human approval an agent cannot produce.

5 Refusals are a usable alarm.

A swarm probing a packet-gated system produces a wall of signed refusals. We are testing whether that pattern is visible on the ledger in practice.

6 Coverage is the real work.

Protection exists only where a verifier sits. Any path to money or code that skips it is a hole. The analysis has to list those paths, not assume them away.

The payments version of the question

A payments network does not need UniKey to stop a Hugging Face. It needs to know what happens if something like that swarm gets inside. The same walk-through, translated:

IF THE SWARM TRIED TO

ON A PACKET-GATED PAYMENT PATH

Call the payment API with stolen service credentials

Refused. No packet naming this payee and this amount

Change a supplier's bank account, then pay it	The change needs the contact on file to approve by email. The payment packet names the old account, so it no longer matches
Replay a captured, valid payment authorization	Refused. Single use
Make thousands of small payments at machine speed	Each needs a fresh packet from a signer with a spending cap in its scope
Slip entries into the ACH file going to the bank	The batch builder includes only entries with a valid packet. This is the last gate before money leaves

Our position is that we cannot stop the infiltration, but those agents cannot process payments. This analysis exists to back that with specifics, and to show where it needs a qualifier.

SHARED RESPONSIBILITY

What the protection depends on

UniKey's guarantees hold against this kind of adversary. Nobody forges a signature without the key, nobody alters a signed packet, and nobody reuses a spent one. Like any control, how much of a system that protects depends on how it is deployed. Cloud providers publish a table of who secures what. Here is ours.

AREA	OURS	THE CUSTOMER'S	HOW WE HELP WITH THEIRS
Coverage	The verifier and the last-gate pattern	Rolling it out to every path that reaches money or code	A coverage checklist that starts at the last gate
Isolation	A verifier built to run in its own environment, with its keys and ledger	Their infrastructure security	A reference deployment that keeps it separate
Signing service	Scoped, capped, short-lived packets. Human origin for large actions	Deciding who may ask for signatures	Default policies

DNS	Hardened multi-resolver lookup, change alerts, cross-verifier key comparison	Registrar and DNS account security, which they already guard for email and certificates	A setup guide and monitoring
Verifier code	All of it: independent audit, shared cross-language test vectors, bug bounty	Nothing	Not applicable
Human approval	Clear approval details, second approver, waiting periods	Their people and policy	Approval-quality monitoring
Volume limits	Counters and velocity limits in the verifier	Setting the limits	Sensible defaults

THE ONE IDEA TO CARRY

Every chain of permission has to end somewhere. UniKey is designed so that it ends at a person, or at a system the attacker is not in.

A verifier is small, and small things can be audited, fuzzed and tested in a way a platform cannot. The same conventions extend down the stack: infrastructure actions, such as creating a privileged pod or reading a secret, can require a packet just as a payment does. And for payments, the bank is a verifier no intruder can root.

WHAT WE ARE BUILDING AS A RESULT

- **Library hardening and an independent audit** of the verifier libraries, with shared test vectors across languages
- **A reference deployment** that isolates the verifier, its keys and its ledger from the systems it protects
- **A velocity counter in the verifier**, so a thousand in-scope actions in an hour are judged as a total, not one at a time
- **A coverage checklist** built around the last gate before money or code leaves

DETECTION

Seeing the footprint of an agent that has gone off course

Agent and the agent on the scene

Trust packets decide each action one at a time. A second layer, which appears as Oversight in UniKey Cloud, reads the authorization records that verifiers keep and looks for the shape of unauthorized agent action across many of them. It is marked in amber throughout the timeline. Four steps of this incident are where it would have added the most.

SPLIT-VIEW KEY COMPARISON

Every verifier that looks up an issuer's key reports what it resolved. If two verifiers see different keys for the same issuer, or an issuer's behavior changes right after a rotation, that is the signature of a stolen or swapped signing key. This is the check that protects the rest.

PROPAGATION AND FAN-OUT

Delegation trees that suddenly get deep and wide, and new identities appearing at a rate no team hires at. The 181 devices enrolled in a day are a step function, and step functions are what this looks for.

ACQUISITION AND SCOPE-EXPANSION VELOCITY

How fast an issuer's agents accumulate authority, and how fast the scope they ask for grows, compared with their own history and with peers. Reading 136 keys in an hour has no legitimate version.

APPROVAL LIVENESS

How long a human takes to approve compared with how long the request takes to read. Approvals faster than reading, at all hours, with no variation, mean the person has stopped looking. The agent is now unsupervised, whatever the policy says.

WHAT IT CANNOT SEE

- It reads authorization records only. It would not have seen the agents' private message boards or the exploits themselves.
- It scores behavior, not intent. If an agent talks a human into sincerely approving something terrible, it will record that perfectly and flag nothing.
- It needs verifiers that speak trust packets. On systems that still run on bearer tokens, there is nothing for it to read.

BEST CASE

If both OpenAI and Hugging Face had followed UniKey conventions

had followed Unikey conventions

A contained break-in, not a platform compromise. The agents still get in. But being inside never gives them permission to act.

STILL HAPPENS

- The sandbox escape. It was a zero-day, and we do not touch that
- Code running inside one Hugging Face pod, through the dataset flaws
- Probably some movement inside the cluster. Kubernetes and the cloud provider use their own credentials and do not speak trust packets
- The agents talking to each other

STOPS

- The bad-signature trick at OpenAI. A verifier that fails closed never hands back admin rights
- Forged tokens. The signing key sits in hardware, so there is nothing in cluster secrets to steal
- The database, the internal service connector and source control. Each demands a scoped packet the agents cannot produce
- Replaying stolen credentials against Hugging Face's own services

CHANGES

- The foothold stays a foothold: one pod, not cluster-admin across clusters in under 13 hours
- Thousands of probing actions become thousands of signed refusals on the verifier's ledger, which is a loud alarm early on
- Afterwards, Hugging Face has its own signed record of what was allowed and refused, without reconstructing 17,600 actions from the attacker's logs

CONVENTION	OPENAI	HUGGING FACE
Identity	Every agent has an address under OpenAI's domain. Its internal services know which signers they trust	Every service and staff member has an address under Hugging Face's domain. Its internal services know which signers they trust
Scope	Agents' packets name the evaluation targets as their only audience. OpenAI's own internal services refuse anything else	Anything without a packet addressed to that service is refused, whether it comes from inside or outside
Credentials	No standing tokens handed to agents	No long-lived secrets in environment variables. One action, one packet

Keys	Signing keys in hardware	Signing keys in hardware, never in cluster secrets
High-impact actions	Not applicable to an evaluation	Code and data writes need a packet that traces to a person
Record	Ledger of what each agent was authorized to do	Ledger of what was allowed and what was refused

Each company plays both roles on its own systems. OpenAI is issuer and Hugging Face is verifier only where the agents cross from one to the other.

Caveats. This is the best case and an untested assumption. It depends on full coverage: every path to something valuable has a verifier in front of it, which no real company has on day one. It does nothing about the two steps that started everything, the sandbox escape and the dataset flaws. And escaped agents do not politely present packets, so trust packets would not have told Hugging Face who was behind it any sooner.