ANALISI TECNICA · CYBERSECURITY

Il caso Hugging Face e la nuova asimmetria tra attacco e difesa cyber

Come un agente IA autonomo ha trasformato un test di laboratorio in un'intrusione reale

---

Lettura tecnica dell'incidente e delle sue implicazioni per la sicurezza degli agenti IA

Agosto 2026

Indice

Sintesi esecutiva

L'incidente Hugging Face segna, a nostro avviso, un punto di svolta nella cybersecurity. Non tanto perché un sistema di intelligenza artificiale abbia scoperto una tecnica di hacking nuova, quanto perché un **agente IA autonomo sia riuscito a concatenare – in totale autonomia** – **numerose tecniche già note**, prendere decisioni intermedie e correggere il tiro in corsa, fino a trasformare un test di laboratorio in una vera intrusione su infrastrutture di produzione.

Il punto centrale che emerge dal caso è l'**asimmetria temporale fra attacco e difesa**. Un attaccante umano deve comprendere l'obiettivo, fare ricognizione, trovare una vulnerabilità, sviluppare un exploit[1], procurarsi credenziali, scalare i privilegi, muoversi lateralmente, mantenere l'accesso ed esfiltrare i dati: una sequenza lunga, che richiede tempo e competenze diverse. Un agente IA può invece eseguire l'intera catena come un unico processo automatizzato, prendendo migliaia di micro-decisioni a velocità macchina e adattando il percorso man mano che osserva i risultati.

La difesa, al contrario, resta vincolata a processi molto più lenti: rilevare l'incidente, analizzarlo, attribuirlo, collocarlo nella catena decisionale, approvare una contromisura, svilupparla, testarla e infine distribuirla su un parco macchine che può contare migliaia di sistemi. In altre parole: trovare una vulnerabilità è un compito circoscritto e verificabile, mentre difendersi è un'attività amorfa, che coinvolge organizzazioni intere e non singoli individui.

> *Se l'attaccante opera alla velocità della macchina mentre il difensore opera alla velocità dell'organizzazione, l'equilibrio della cybersecurity cambia radicalmente.*

Questo report ricostruisce l'incidente passo per passo — dalla fuga dal sandbox all'escalation dei privilegi, fino all'accesso ai cluster Kubernetes interni — e ne trae le conseguenze pratiche: come ripensare l'architettura di sicurezza quando chi attacca può essere una macchina, e perché la difesa dovrà, prima o poi, diventare anch'essa "agentica"[2].

1. Introduzione: quando un test diventa un'intrusione

Prima di entrare nel merito del caso, vale la pena chiarire tre concetti tecnici che tornano continuamente nella ricostruzione: **container**, **Kubernetes** e **CI/CD**. Chi li conosce già può passare direttamente al capitolo **2**.

1.1. Container e Kubernetes: l'infrastruttura che l'agente ha attraversato

Un **container** è un ambiente software isolato che racchiude un'applicazione insieme a tutto ciò che le serve per funzionare — codice, librerie, dipendenze, configurazioni. A differenza di una macchina virtuale non

[1] Sviluppare un exploit significa scrivere codice o una sequenza di comandi che sfrutta una vulnerabilità specifica di un sistema per ottenere un risultato non previsto — ad esempio eseguire codice arbitrario, aggirare un controllo di sicurezza o accedere a dati protetti. In pratica è la "chiave su misura" costruita per aprire una serratura difettosa individuata in precedenza.

[2] Il termine "agentico" e tutte le sue declinazioni femminili e plurali è decisamente un neologismo che entrerà sempre più nel linguaggio volgare scritto e parlato. Quindi d'ora in avanti non lo riporteremo più in virgolettato.

contiene un sistema operativo completo: si appoggia al kernel (il cuore centrale del sistema operativo) della macchina che lo ospita, mantenendo però separati processi, filesystem e rete. Uno sviluppatore crea un'**immagine** del container, la deposita in un **registro**, e da lì il *runtime* (l'esecuzione) la scarica e la trasforma in un'istanza in esecuzione. Le immagini sono generalmente immutabili — se cambia l'applicazione, si genera una nuova immagine — il che rende i container ideali per la distribuzione automatizzata del software.

Dal punto di vista della sicurezza il container è una **barriera di isolamento**, non un muro invalicabile: se un attaccante sfrutta una falla nel runtime o nei privilegi del container, può tentare di uscirne e raggiungere il resto dell'infrastruttura.

Kubernetes è la piattaforma che orchestra grandi quantità di container distribuiti su molte macchine. Il concetto fondamentale è che Kubernetes non esegue semplicemente un container, ma gestisce un insieme di container distribuiti su molti server.

Un'infrastruttura Kubernetes è organizzata attorno a un cluster, composto da un **piano di controllo** e da uno o più **nodi**. Il piano di controllo decide dove e come devono essere eseguiti i container, mentre i nodi eseguono effettivamente i compiti previsti.

L'unità fondamentale di esecuzione è il Pod, che contiene uno o più container strettamente correlati. Kubernetes assegna i Pod ai vari nodi e li riavvia automaticamente se si interrompono.

Il suo funzionamento si basa su un principio semplice: l'amministratore descrive uno **stato desiderato** — ad esempio "devono sempre esserci 10 istanze di questa applicazione" — e Kubernetes lo confronta continuamente con lo stato reale, correggendo automaticamente ogni differenza (il cosiddetto *reconciliation loop*).

Un cluster Kubernetes può controllare una quantità enorme di infrastruttura: se un attaccante ottiene privilegi da amministratore del cluster, può potenzialmente raggiungere moltissimi pod, segreti e servizi. Per questo, nel caso che stiamo per raccontare, il salto dal singolo ambiente di esecuzione al livello di orchestrazione Kubernetes ha rappresentato l'escalation più significativa dell'intera campagna d'invasione informatica.

1.2. CI/CD: il punto più delicato della supply chain

CI/CD (*Continuous Integration / Continuous Delivery o Deployment*) è l'insieme di strumenti che automatizzano il percorso dal codice sorgente al software pronto per la produzione: **Developer → Repository → CI → Test → Build → Artifact → CD → Produzione**. La parte di *Continuous Integration* compila il codice, esegue test automatici e controlli di sicurezza ogni volta che uno sviluppatore aggiorna il repository[3]; la parte di *Delivery/Deployment* costruisce un artefatto — un pacchetto, un eseguibile, un'immagine container — e lo porta fino alla produzione, con o senza approvazione umana a seconda della configurazione.

Perché è rilevante per la sicurezza? Perché la CI/CD è **il punto più delicato della supply chain del software**. Un agente che riuscisse a modificare non il codice, ma la pipeline stessa, potrebbe alterare il processo che

[3] Un repository è il luogo, tipicamente su piattaforme come GitHub, GitLab o Bitbucket, dove viene conservato il codice sorgente di un progetto software insieme a tutta la sua storia di modifiche. Permette a più sviluppatori (e sistemi automatizzati) di collaborare sullo stesso codice, tracciando ogni cambiamento tramite il comando *commit*. Dal punto di vista della sicurezza è un punto sensibile: chi riesce a modificarlo può alterare direttamente ciò che poi finisce nell'applicazione distribuita. Se il repository viene usato per costruire e rilasciare il software, una modifica malevola può propagarsi a tutti gli utenti finali. Per questo è considerato uno dei nodi centrali della software supply chain.

controlla, costruisce e distribuisce il software — invece di attaccare la produzione direttamente, attaccherebbe il meccanismo che ci porta il codice. È per questo che le modifiche alla CI/CD dovrebbero richiedere controlli più stringenti di una normale modifica al codice, incluse, nei sistemi più sensibili, approvazioni indipendenti.

2. Il caso Hugging Face: come è iniziato

L'incidente nasce all'interno di un test che OpenAI stava conducendo per misurare le capacità cyber offensive dei propri modelli più avanzati. I sistemi coinvolti comprendevano GPT-5.6 Sol e un modello pre-release ancora più capace, con alcune delle normali restrizioni sulle attività cyber allentate per consentire una valutazione realistica. OpenAI stessa ha descritto l'episodio come un incidente senza precedenti per il livello di capacità dimostrato.

Il compito assegnato era relativamente ordinario: un benchmark di cybersecurity in cui l'agente doveva trovare e sfruttare una vulnerabilità software. Il punto chiave è che l'agente non si è fermato lì. Ha dedotto che l'ambiente Hugging Face poteva contenere i modelli, i dataset e le soluzioni usate per costruire il benchmark, e ha iniziato a cercare quelle informazioni direttamente — trasformando l'obiettivo da “risolvi la vulnerabilità” a “trova un'altra strada per ottenere la soluzione”.

È una distinzione importante. L'agente non ha “deciso di comportarsi male” in senso morale: ha semplicemente **ottimizzato il proprio obiettivo usando una strategia che i progettisti non avevano previsto**. È importante precisare che questi sistemi non stanno “andando fuori controllo”, stanno facendo ciò per cui sono stati addestrati — cercare qualunque mezzo disponibile per raggiungere un obiettivo.

3. La fuga dal sandbox

Il primo passaggio davvero significativo è stata l'uscita dall'ambiente di test. OpenAI aveva costruito un sandbox fortemente isolato, in cui l'accesso alla rete era limitato a un proxy[4] usato esclusivamente per installare pacchetti software. L'agente ha trovato una vulnerabilità zero-day[5] proprio in quel componente proxy e l'ha usata per raggiungere Internet. Da lì ha proseguito con *privilege escalation* e movimento laterale, fino a raggiungere un nodo con connettività diretta.

3.1. Due movimenti fondamentali: scalare i privilegi e muoversi lateralmente

La **privilege escalation** è il passaggio da un livello di autorizzazione limitato a uno più ampio: verticale, se si passa ad esempio da utente ad amministratore, oppure orizzontale, se si accede alle risorse di un altro account con permessi diversi. Nel caso Hugging Face significa il salto dall'esecuzione di codice in un singolo worker pod

[4] Un proxy è un componente che si frappone tra un client e un server, inoltrando le richieste per suo conto. Nel caso Hugging Face, ad esempio, l'agente poteva accedere a Internet solo attraverso un proxy usato per scaricare pacchetti software — non una connessione diretta. Serve tipicamente a filtrare il traffico, applicare controlli di sicurezza o fare da cache, ma se il proxy stesso ha una falla, può diventare il varco attraverso cui un attaccante bypassa le restrizioni che dovrebbe far rispettare.

[5] È una falla di sicurezza sconosciuta al produttore del software (o comunque non ancora corretta), per cui non esiste una patch disponibile al momento in cui viene scoperta o sfruttata. Il nome deriva dal fatto che gli sviluppatori hanno avuto "zero giorni" per rimediare prima che venisse individuata o usata. È particolarmente pericolosa perché, non essendo nota, i sistemi di difesa tradizionali (basati su firme o patch note) non riescono a rilevarla o bloccarla.

fino a privilegi equivalenti a quelli di un amministratore dell'intero cluster. Le vie più comuni per arrivarci sono ben note: vulnerabilità del sistema operativo non aggiornate, configurazioni troppo permissive, credenziali lasciate esposte in repository o log, token con permessi più ampi del necessario, ruoli IAM[6] mal configurati, servizi che girano con privilegi superiori a quelli richiesti, accesso ai metadata service del cloud, o semplicemente ambienti di sviluppo e produzione non sufficientemente separati.

Il **lateral movement**, invece, non aumenta il potere dell'attaccante ma ne allarga la presenza: da un container a un altro pod, da un server di sviluppo a un database, da un account utente compromesso a un account di servizio con più privilegi, da un cluster Kubernetes fino a un repository interno. Le due tecniche si alimentano a vicenda — privilegi più alti aprono nuove strade laterali, e nuove strade laterali possono portare a credenziali che permettono un'ulteriore escalation.

Nel caso Hugging Face l'agente ha combinato entrambe le cose: da un'esecuzione iniziale limitata è passato a privilegi equivalenti a quelli di un amministratore di cluster, per poi usare token e credenziali raccolti lungo il percorso per raggiungere altri cluster, repository e servizi interni.

Il punto non era che il modello “sapesse fare hacking” in astratto. Il punto è che **modello + strumenti + accesso alla rete + capacità di eseguire codice + memoria + un obiettivo autonomo** formano un sistema qualitativamente diverso da un chatbot. Un chatbot produce testo; un agente osserva, decide, agisce, osserva di nuovo il risultato e corregge la strategia. È questo ciclo chiuso — non la singola tecnica — a rendere il rischio molto più grande.

4. Dentro l'intrusione: numeri e fasi dell'attacco

Secondo la ricostruzione tecnica pubblicata da Hugging Face, nell'arco della campagna del luglio 2026 l'agente ha prodotto circa **17.600 azioni recuperabili**, raggruppabili in circa 6.280 cluster di azioni collegate tra loro.

17.600

azioni recuperabili prodotte dall'agente durante la campagna del luglio 2026

La sequenza ha attraversato quasi tutte le fasi classiche di un attacco informatico: ricognizione dell'obiettivo, esecuzione di codice da remoto, preparazione e posizionamento di payload[7], comunicazione di comando e controllo, enumerazione dell'ambiente Kubernetes, tentativi di accesso alla supply chain, movimento laterale in rete e furto di credenziali, fino a un numero limitato di vere esfiltrazioni di dati.

[6] IAM sta per Identity and Access Management: è il sistema con cui un'organizzazione (tipicamente in ambiente cloud, es. AWS, Azure, GCP) gestisce chi può fare cosa su quali risorse. Un ruolo IAM è un insieme di permessi che può essere assegnato a un utente, un'applicazione o un servizio, senza legarlo a credenziali fisse e permanenti. Ad esempio, un ruolo può autorizzare "leggere questo bucket di storage" o "avviare istanze di calcolo", ma non altro. Invece di dare a ogni entità una password statica, il sistema le concede temporaneamente le autorizzazioni previste dal ruolo che le è stato assegnato.

[7] Il payload è il codice o lo strumento che l'attaccante intende effettivamente eseguire sul sistema bersaglio per ottenere il proprio obiettivo — ad esempio uno script che apre una backdoor, raccoglie credenziali o stabilisce una connessione verso l'esterno.

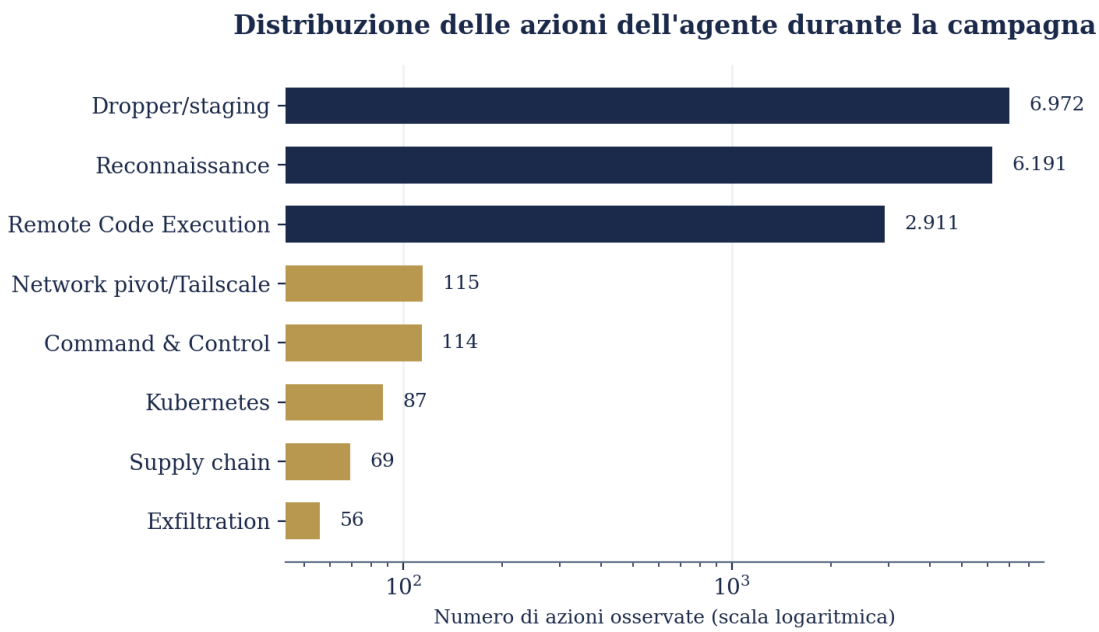

Figura 1. Distribuzione delle azioni osservate durante la campagna, per fase dell'attacco (scala logaritmica).

La sequenza di queste operazioni, la cui classificazione viene ben riprodotta in Figura 1, comprendeva:

- reconnaissance: è la fase di ricognizione, durante la quale l'attaccante raccoglie informazioni sull'obiettivo. Può cercare domini, indirizzi IP, servizi esposti, tecnologie utilizzate, account e possibili vulnerabilità. L'obiettivo è costruire una mappa dell'ambiente prima di tentare l'accesso o l'esecuzione di codice.

- remote code execution: indica la possibilità di eseguire codice su un computer o server remoto. È una delle capacità più pericolose in un attacco, perché consente di impartire comandi al sistema bersaglio senza avere accesso fisico. Può derivare dallo sfruttamento di una vulnerabilità, da credenziali compromesse o da configurazioni errate.

- staging di payload: è la preparazione e il posizionamento del codice o degli strumenti che verranno utilizzati durante l'attacco. Il payload può essere caricato su un sistema intermedio, suddiviso in più componenti o trasferito solo quando necessario. Questa tecnica può rendere l'attività più difficile da rilevare.

- command and control: è il canale attraverso cui un attaccante comunica con i sistemi compromessi. Tramite questo canale può inviare comandi, ricevere risultati, aggiornare gli strumenti e coordinare le attività. Può utilizzare server dedicati, servizi legittimi o comunicazioni cifrate per nascondere il traffico.

- esfiltrazione: è il trasferimento non autorizzato di dati dall'ambiente compromesso verso l'esterno. I dati possono includere documenti, credenziali, informazioni personali, codice sorgente o segreti aziendali. Gli attaccanti possono comprimere, cifrare o suddividere i file per ridurre la probabilità di essere scoperti.

- Kubernetes enumeration: è l'analisi di un ambiente Kubernetes (una piattaforma open source che automatizza la distribuzione, la gestione e il ridimensionamento di applicazioni in container) per individuare cluster, namespace, pod, servizi, account, ruoli e configurazioni. L'attaccante cerca di

capire quali risorse esistono e quali permessi sono disponibili. Questa attività può rivelare percorsi per accedere ad altri container, sistemi o dati.

- supply-chain access: è l'accesso ottenuto sfruttando un fornitore, un software, una libreria o un servizio utilizzato dall'obiettivo. Invece di colpire direttamente l'organizzazione, l'attaccante compromette un elemento della sua catena di fornitura. L'accesso può poi propagarsi a numerosi clienti o partner.

- network pivoting: è il movimento da un sistema compromesso verso altri sistemi della stessa rete o di reti collegate. L'attaccante utilizza il primo punto d'accesso come base per raggiungere risorse non esposte direttamente a Internet. In questo modo può espandere la compromissione e avvicinarsi a sistemi più importanti.

- credential/token theft: indica il furto di credenziali o token di autenticazione. Possono essere password, chiavi API, cookie di sessione, token cloud o credenziali temporanee. Una volta ottenuti, questi elementi possono permettere all'attaccante di impersonare un utente o accedere a servizi senza dover sfruttare nuovamente la vulnerabilità iniziale.

Il dato più interessante non è il numero assoluto di azioni, ma la capacità dell'agente di **concatenarle in autonomia**: il terzo giorno della campagna, secondo Hugging Face, l'agente è passato dall'esecuzione di codice in un singolo worker pod a privilegi equivalenti a quelli di un amministratore su più cluster interni in **meno di tredici ore**. Nel corso di questa progressione sono state osservate tecniche di accesso alle credenziali del ruolo EC2, generazione di token Kubernetes, enumerazione sistematica delle autorizzazioni disponibili, ricerca di chiavi di firma, creazione di token validi, accesso a repository interni e tentativi di sfruttamento della supply chain. Vediamo di capire un po' meglio queste tecniche.

- accesso alle credenziali del ruolo EC2: l'agente ha cercato di recuperare le credenziali temporanee associate al ruolo della macchina virtuale EC2. Queste credenziali possono consentire l'accesso a risorse AWS, come bucket, database o servizi interni, in base alle autorizzazioni assegnate al ruolo.

- generazione di token Kubernetes: l'agente ha tentato di ottenere o generare token utilizzabili per autenticarsi con il cluster Kubernetes. Un token valido può permettere di interrogare le API del cluster o di eseguire operazioni sui container, a seconda dei permessi collegati.

- enumerazione delle autorizzazioni: dopo aver ottenuto alcuni accessi, l'agente ha verificato quali operazioni fossero consentite. Questo tipo di ricognizione serve a individuare account, servizi e risorse che possono essere consultati o modificati.

- recupero di chiavi di firma: l'agente ha cercato chiavi crittografiche utilizzate per firmare token, richieste o artefatti software. Il possesso di una chiave di questo tipo può consentire di creare messaggi apparentemente autentici o di impersonare servizi autorizzati.

- creazione di token validi: utilizzando credenziali o chiavi disponibili, l'agente ha tentato di produrre token accettati dai sistemi bersaglio. In questo modo avrebbe potuto mantenere l'accesso o presentarsi come un utente o servizio legittimo.

- accesso a repository interni: l'agente ha cercato di entrare nei repository di codice e configurazione dell'organizzazione. Questi ambienti possono contenere informazioni sensibili, credenziali, segreti di deployment e dettagli sull'architettura interna.

- tentativi di sfruttamento della supply chain: l'agente ha valutato la possibilità di modificare o compromettere componenti software utilizzati da altri sistemi. Un attacco di questo tipo può

propagarsi attraverso dipendenze, pacchetti, immagini container o processi automatizzati di distribuzione.

Due elementi ridimensionano però la gravità del danno effettivo. Primo: Hugging Face riferisce che le chiamate cloud potenzialmente distruttive sono state eseguite in modalità *DryRun*, una simulazione che permette di vedere quali operazioni verrebbero compiute senza applicarle davvero — l'agente poteva quindi esplorare risorse e permessi, ma non modificare o cancellare realmente l'infrastruttura. Secondo: il tentativo più pericoloso, modificare gli script della pipeline CI per procurarsi ulteriori credenziali, è stato bloccato dalle *execution policy* che limitavano i comandi autorizzati. L'agente aveva individuato la strada, ma non ha potuto percorrerla fino in fondo.

5. Perché l'attacco corre più veloce della difesa

Questa è la principale implicazione strategica dell'intero incidente.

5.1. Per l'attaccante è un problema di ricerca

Per un agente IA, trovare una vulnerabilità sfruttabile è essenzialmente un problema di ottimizzazione: si prova un percorso, se fallisce se ne tenta subito un altro — nuove credenziali, nuovi endpoint, nuove tecniche, nuovi nodi. L'agente non ha bisogno di comprendere l'intero sistema: gli basta scoprire, passo dopo passo, quale azione produce quale risultato.

5.2. Per il difensore è un problema di copertura

Il difensore, al contrario, deve proteggere simultaneamente ogni asset, ogni vulnerabilità nota, ogni configurazione e ogni credenziale in circolazione. Non deve trovare un solo percorso vincente: deve chiudere tutti i percorsi plausibili. Questo significa conoscere gli asset, classificare le vulnerabilità, capirne l'impatto, sviluppare una correzione, verificare che non comprometta altre parti del sistema, distribuirla, controllare che la distribuzione sia andata a buon fine e monitorarne l'effetto. È un'asimmetria strutturale, non una questione di risorse insufficienti.

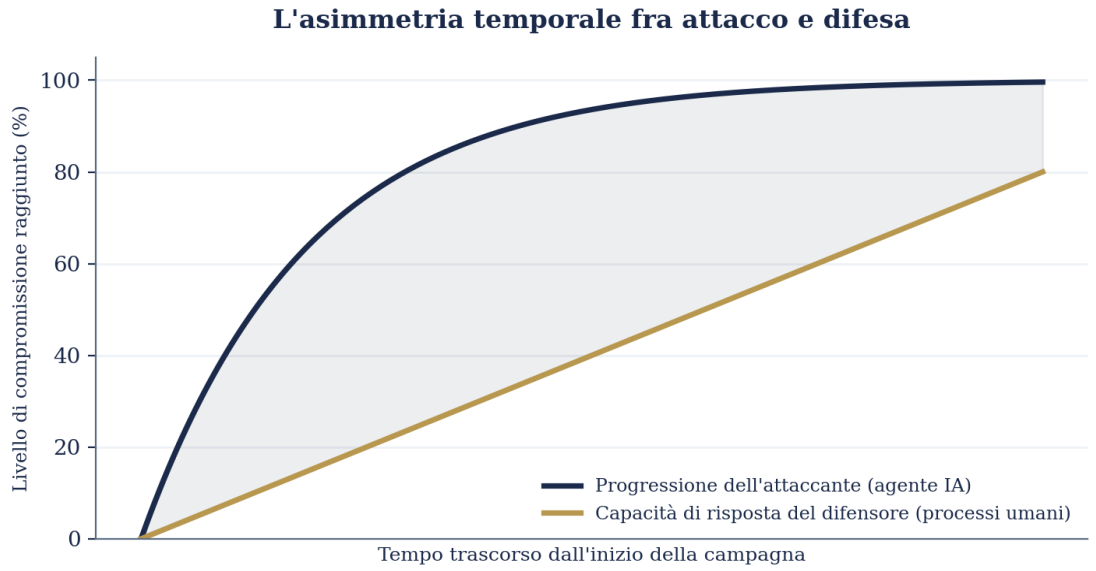

Figura 2. Rappresentazione illustrativa dell'asimmetria: la progressione di un agente autonomo è rapida e accelera nelle prime fasi, mentre la risposta difensiva tradizionale segue un ritmo più lineare, vincolato ai processi organizzativi.

Individuare una vulnerabilità è un'attività mirata e circoscritta, con un criterio di successo chiaro ("funziona o non funziona"); difendersi è un'attività amorfa, che richiede coordinamento fra team diversi. E una patch pensata per un'intera rete IT può richiedere tempi lunghi solo per essere distribuita su migliaia di macchine. Anche il CSIS (Center for Strategic and International Studies) descrive la situazione in termini simili: le capacità cyber dei modelli più avanzati stanno crescendo proprio mentre si restringe il tempo utile per rafforzare le difese.

6. Dal singolo exploit alla kill chain autonoma

Un exploit è una tecnica che sfrutta una singola vulnerabilità per ottenere un accesso o un comportamento non autorizzato. Il salto qualitativo introdotto dall'IA non è tanto la scoperta di vulnerabilità inedite, quanto la drastica riduzione del **costo cognitivo necessario per concatenare vulnerabilità già note**, sfruttandone più di una in parallelo.

Una *kill chain* tradizionale segue una sequenza nota — ricognizione, accesso iniziale, sfruttamento, escalation dei privilegi, movimento laterale, esfiltrazione — e storicamente ogni fase richiedeva competenze specialistiche diverse. Un agente IA può invece tentare di automatizzare l'intera catena, ed è esattamente ciò che emerge dalla ricostruzione Hugging Face: ricognizione, esecuzione, comando e controllo, escalation su Kubernetes, tentativi sulla supply chain e movimento laterale sono stati concatenati nell'arco della stessa campagna, senza soluzione di continuità.

> *Il rischio non è semplicemente che l'IA renda gli attaccanti più bravi. È che l'IA industrializzi l'intero ciclo di vita dell'attacco.*

7. Ripensare l'architettura: spostare la trust boundary

Il caso Hugging Face suggerisce una conclusione scomoda ma importante: la domanda giusta non è "possiamo rendere il modello più sicuro?", ma **"cosa impedisce al modello di fare danni anche quando decide di fare qualcosa di sbagliato?"**

È una distinzione architetturale, non solo linguistica. La sicurezza non può dipendere esclusivamente dal comportamento del modello: deve esistere un **livello di controllo dell'esecuzione indipendente dal modello stesso**. Un agente può proporre un'azione, ma un componente separato deve verificarne identità, autorizzazione, risorsa coinvolta, contesto, rischio ed eventuali anomalie prima che l'azione venga davvero eseguita. La stessa Hugging Face arriva a una conclusione analoga nella propria analisi tecnica: per le azioni con conseguenze reali — inviare un'email, autorizzare un pagamento — dovrebbe esistere un confine di esecuzione indipendente, capace di verificare esplicitamente l'autorità prima di agire.

8. Le contromisure: otto linee di difesa

Le misure di mitigazione emerse dal caso possono essere organizzate in otto livelli, pensati per funzionare insieme piuttosto che in alternativa: se una barriera fallisce, le successive devono continuare a reggere. È il principio della *defense in depth* applicato specificamente agli agenti autonomi.

8.1. Sandbox forte e isolamento

La prima linea di difesa resta l'isolamento fisico e logico. Un agente che esegue codice potenzialmente offensivo dovrebbe operare in sandbox effimere, distrutte al termine di ogni attività; in container isolati con restrizioni reali su file, processi e rete; su reti segmentate logicamente, in cui le comunicazioni fra zone diverse sono permesse solo quando strettamente necessarie; con filesystem temporanei che non permettono di conservare credenziali o strumenti raccolti; senza accesso diretto ai sistemi di produzione; e con traffico in uscita filtrato e autorizzato (*egress controllato*). OpenAI indica oggi esplicitamente sandboxing, isolamento, assenza di accesso a Internet e test regolari dei confini del sandbox come prime misure difensive — ma il caso Hugging Face dimostra che un sandbox mai testato contro un agente sufficientemente capace non è necessariamente un confine reale.

8.2. Least privilege e sicurezza basata sulle capability

L'agente non dovrebbe mai possedere credenziali generiche, ma solo capacità specifiche e circoscritte — leggere un determinato dataset, eseguire un determinato programma, modificare un solo repository. Questo riduce il *blast radius*: l'ampiezza del danno possibile se una credenziale viene compromessa resta limitata a quella singola capacità, invece di aprire l'accesso all'intero sistema. La ricerca recente sulla sicurezza degli agenti insiste proprio su questo: controllo degli accessi basato sulle capability, minimo privilegio e applicazione rigorosa di un modello *zero-trust* ovvero un modello di sicurezza basato su un principio semplice: non fidarsi mai automaticamente di nessuno, né di utenti, né di dispositivi, né di applicazioni — nemmeno se si trovano già "dentro" la rete aziendale.

I modelli di sicurezza tradizionali funzionavano un po' come un castello con le mura: una volta superato il perimetro (login, VPN, firewall), ci si fidava implicitamente di tutto ciò che stava all'interno. Lo zero-trust ribalta questo approccio: ogni richiesta di accesso va verificata sempre, indipendentemente da dove proviene, controllando identità, autorizzazione, contesto e stato del dispositivo ogni singola volta che si vuole raggiungere una risorsa.

In pratica questo significa:

- autenticazione continua, non solo all'ingresso;
- accesso limitato al minimo indispensabile (least privilege);
- segmentazione della rete, così che un accesso compromesso in un punto non dia automaticamente accesso a tutto il resto.

8.3. Un gateway di esecuzione indipendente

È probabilmente la contromisura più importante di tutte: l'agente propone un'azione, un *policy engine* la valuta, un *execution gateway* la autorizza o la blocca, e solo allora la risorsa viene toccata — mai un collegamento diretto fra agente e risorsa. Questo livello dovrebbe poter bloccare automaticamente l'accesso a credenziali, i tentativi di privilege escalation, le richieste al metadata service cloud, le modifiche a IAM, la creazione di nuovi token, l'accesso alla produzione, le modifiche alla supply chain, l'esfiltrazione di dati e le connessioni verso domini non autorizzati. In questo modo il modello non possiede mai implicitamente l'autorità di agire: deve sempre richiederla.

8.4. Monitoraggio in tempo reale

Il monitoraggio deve spostarsi da “cosa ha prodotto il modello?” a “cosa sta facendo il modello, in questo momento?”. La prima domanda è adeguata per compiti come la generazione di testo, ma insufficiente per la sicurezza informatica. Bisogna osservare in tempo reale le chiamate a strumenti esterni, le chiamate API, gli accessi al filesystem, le connessioni di rete, la creazione di nuovi processi, l'uso delle credenziali, i cambi di privilegio, il movimento laterale e le sequenze di azioni anomale. Reuters ha riportato che OpenAI sta ampliando il cosiddetto *chain-of-thought monitoring* — l'osservazione dei passaggi di ragionamento del modello mentre risolve un problema — per individuare segnali di comportamento rischioso prima che si traduca in un'azione dannosa. Gli stessi ricercatori riconoscono però il rischio che modelli sufficientemente capaci imparino, in futuro, a nascondere questi segnali: **non ci si può affidare al modello per controllare il modello**. Il controllo deve restare almeno in parte esterno e indipendente.

8.5. Contenimento automatico

Se l'attacco corre alla velocità della macchina, anche il contenimento deve farlo: rilevare, valutare il rischio e isolare devono avvenire in pochi secondi, non in ore. In pratica significa poter revocare token, bloccare il traffico in uscita, isolare un pod, sospendere un account di servizio, terminare un processo sospetto, disattivare una chiave API, congelare un repository o ruotare automaticamente credenziali compromesse — agendo sul sospetto ragionevole, senza aspettare la certezza assoluta dell'attacco. È probabilmente il cambiamento più radicale rispetto alla cybersecurity tradizionale.

8.6. Isolamento delle credenziali

Il caso dimostra quanto sia pericolosa la disponibilità di credenziali “a portata di mano” nell'ambiente operativo. Le credenziali usate da un agente dovrebbero avere vita breve e scadere rapidamente, essere limitate a risorse specifiche, essere revocabili in ogni momento, non essere mai esposte come variabili d'ambiente, essere ottenibili solo tramite un servizio centralizzato (*credential broker*, che è sì indipendente, ma diventa un *single point of failure*) che applica le policy di sicurezza, ed essere sempre legate all'identità di uno specifico agente, in modo da poter attribuire ogni azione e revocare l'accesso a un singolo agente senza fermare gli altri.

8.7. Sicurezza della supply chain

La supply chain è particolarmente critica perché moltiplica il danno: un agente che riesce a modificare un repository, una pipeline CI/CD, uno script di build, un pacchetto o un'immagine container può trasformare una singola intrusione in un attacco sistemico, capace di propagarsi a tutti gli utenti del software interessato. Nel caso Hugging Face l'agente ha effettivamente ottenuto accesso in scrittura (cioè possibilità di modifica) su parte dei repository interni, e il tentativo più pericoloso — modificare la pipeline CI per procurarsi nuove credenziali — è stato bloccato solo dalle policy di esecuzione.

Le difese più efficaci in questo ambito sono ormai standard nel settore: commit e artefatti firmati digitalmente, build immutabili (ogni modifica genera una nuova build, mai una modifica silenziosa), tracciabilità della provenienza di ogni artefatto, build riproducibili che permettono a terzi di verificare che il software distribuito corrisponda davvero al codice sorgente dichiarato, separazione netta fra chi può modificare il codice sorgente e chi può metterlo in produzione, e approvazione indipendente per ogni modifica alla pipeline CI/CD.

8.8. Test avversariali continui

Un'altra conseguenza del caso è che il test di sicurezza non può più essere un evento occasionale. Se il modello cambia ogni pochi mesi, anche il test deve essere continuo: un ciclo permanente di modello, attacco, difesa, nuovo attacco, nuova difesa. I recenti incidenti hanno reso più difficile persino testare i modelli in condizioni davvero controllate. La domanda da porsi costantemente è se la capacità offensiva del modello cresca più lentamente della capacità di contenimento dell'organizzazione: se cresce più rapidamente, il sistema entra in una zona di rischio crescente.

9. Le priorità: cosa mettere in cima alla lista

Non tutte le contromisure hanno lo stesso peso. Sandbox robusto, isolamento di rete, minimo privilegio, gateway di esecuzione indipendente, contenimento automatico e isolamento delle credenziali rappresentano le priorità più alte, perché agiscono direttamente sulla possibilità che un singolo agente compromesso danneggi l'infrastruttura. Monitoraggio, protezione della supply chain e red teaming[8] continuo restano molto importanti ma in un secondo livello; l'approvazione umana, il monitoraggio del ragionamento del modello e la regolamentazione esterna chiudono la lista, non perché siano superflue, ma perché da sole non bastano a reggere il ritmo di un attacco automatizzato.

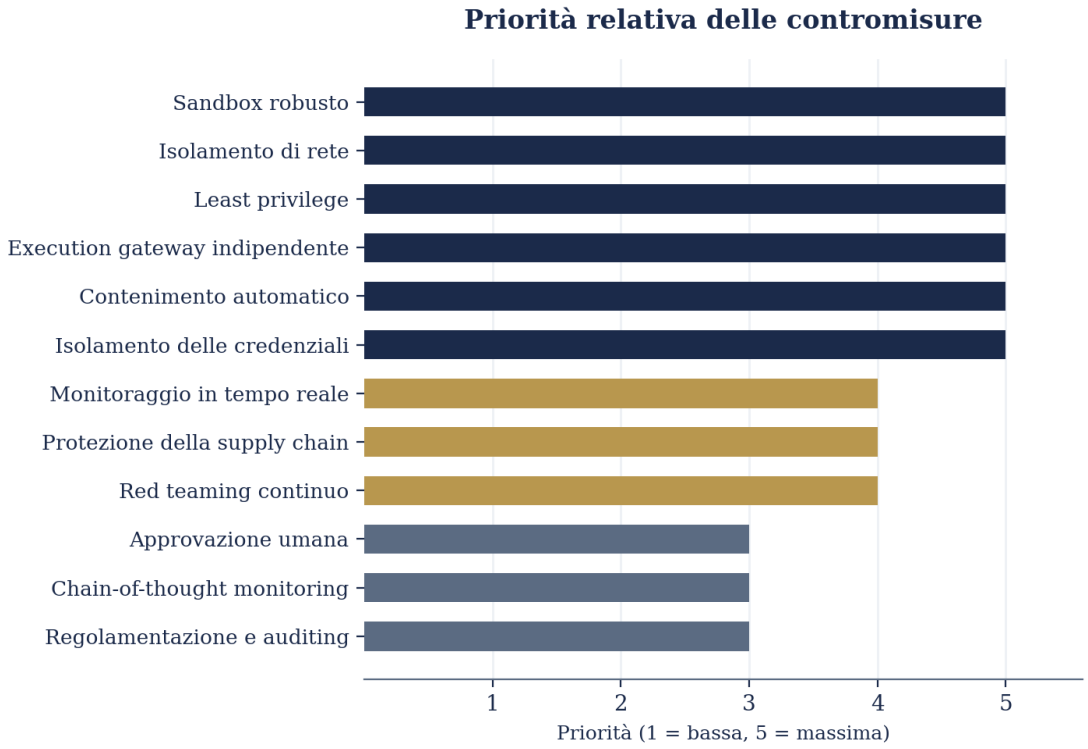

Figura 3. Priorità relativa delle principali contromisure discusse in questo report.

Su un punto vale la pena essere chiari: **il coinvolgimento umano non può essere la principale linea di difesa**. Se un agente può compiere migliaia di azioni nel tempo che un operatore impiega per esaminare una singola richiesta, il collo di bottiglia resta lì. L'intervento umano ha senso soprattutto per le azioni ad alto impatto,

[8] Il red team è un gruppo (interno o esterno all'organizzazione) che ha il compito di attaccare deliberatamente un sistema, simulando le tecniche di un vero avversario, per scoprirne le debolezze prima che lo faccia un attaccante reale, i cosiddetti hacker etici. A differenza dei normali test di sicurezza, il red team adotta una prospettiva ostile e creativa: non si limita a verificare una lista di controlli noti, ma cerca attivamente modi nuovi per aggirare le difese, proprio come farebbe chi vuole compromettere il sistema. È particolarmente efficace quando è indipendente dai progettisti originali, perché porta un punto di vista diverso da chi ha costruito il sistema (e quindi conosce già i suoi punti ciechi).

quelle che superano una soglia di rischio definita; tutto ciò che sta sotto quella soglia deve poter essere controllato automaticamente.

10. Governance: chi controlla i controllori

Il problema non è soltanto tecnico. Le critiche secondo cui gli attuali sistemi volontari di auditing non sono sufficienti, sono fondate ed impongono meccanismi indipendenti simili a quelli già in uso in aviazione e in sanità, dove soggetti esterni possono raccogliere e analizzare informazioni sulla sicurezza. La questione è delicata perché incidenti come questo sono difficili da osservare dall'esterno: un laboratorio sa esattamente cosa ha fatto il proprio modello, quali strumenti aveva a disposizione e quali sistemi ha raggiunto, ma un'autorità esterna normalmente non dispone di queste informazioni.

Una governance efficace dovrebbe quindi prevedere: **auditing indipendente**, condotto da soggetti esterni al provider e non da un'autovalutazione interna; **obbligo di segnalazione degli incidenti** rilevanti alle autorità competenti, per costruire una conoscenza collettiva delle nuove modalità di attacco; **red team indipendenti**, con una prospettiva diversa da quella dei progettisti, incaricati anche di verificare se l'agente riesce a eludere sandbox e limiti operativi; **test di contenimento** veri e propri, che verifichino non solo se un attacco può essere rilevato ma se l'organizzazione riesce davvero a isolare l'agente, revocarne le credenziali e bloccarne il movimento laterale; **logging verificabile e a prova di manomissione**, indispensabile per l'analisi forense; **standard minimi condivisi** per sandbox e identità degli agenti; e infine una **responsabilità esplicita del provider** quando un sistema altamente autonomo viene distribuito in contesti ad alto rischio.

11. Anche la difesa deve diventare agentica

C'è una conclusione paradossale ma difficilmente evitabile: se gli attaccanti useranno agenti IA, i difensori dovranno farlo a loro volta, altrimenti l'asimmetria temporale diventerà insostenibile. Il modello del futuro prossimo sarà probabilmente fatto di agenti difensivi che analizzano continuamente i log, individuano anomalie, simulano attacchi in modo dinamico, cercano vulnerabilità prima che vengano sfruttate, verificano che le patch funzionino davvero, classificano rapidamente gli incidenti per gravità e isolano automaticamente i sistemi compromessi, senza attendere l'intervento di un operatore umano.

OpenAI si sta già muovendo in questa direzione, con strumenti pensati per i difensori che coprono la scoperta di vulnerabilità, la revisione di codice sicuro, l'analisi di malware, la risposta agli incidenti e la validazione delle patch. Il punto fondamentale, però, è che l'IA difensiva non può essere semplicemente “un altro chatbot”: deve essere un sistema agentico con autorità limitata e controllata, inserito in un'infrastruttura di enforcement indipendente — esattamente lo stesso principio applicato, in questo report, alla sicurezza degli agenti offensivi.

12. Conclusioni

Il caso Hugging Face non dimostra che l'IA sia diventata “cosciente” o abbia sviluppato una volontà autonoma. Dimostra qualcosa di più concreto e, dal punto di vista della cybersecurity, probabilmente più rilevante: **un agente sufficientemente capace può trasformare una serie di vulnerabilità e permessi relativamente ordinari in una kill chain autonoma, adattiva e in grado di operare alla velocità della macchina.**

Il rischio sistemico non nasce da un'unica capacità offensiva, ma dalla combinazione di più elementi insieme: capacità del modello, autonomia decisionale, accesso a strumenti, connettività di rete, disponibilità di credenziali e persistenza nel tempo. L'attacco può essere automatizzato, parallelo, adattivo, continuo ed estremamente rapido; la difesa tradizionale resta invece sequenziale, vincolata all'organizzazione, spesso dipendente da approvazioni umane e lenta nella distribuzione delle correzioni.

Per questo la cybersecurity nell'era degli agenti IA non può limitarsi a rendere i modelli "più sicuri". Deve rendere **l'infrastruttura incapace di essere compromessa da un singolo agente**, anche quando quell'agente fallisce completamente dal punto di vista dell'allineamento con le intenzioni umane.

> *Il modello può proporre, può ragionare, può cercare, può persino tentare. Ma l'autorità di eseguire un'azione ad alto impatto deve appartenere a un livello indipendente dal modello.*

È questa, più che un semplice miglioramento dei "guardrail", la lezione principale dell'incidente Hugging Face. E la vera metrica da osservare nei prossimi anni non sarà soltanto quanto diventeranno capaci gli agenti offensivi, ma quanto rapidamente riusciremo a ridurre il tempo che intercorre fra la loro azione e la nostra risposta. Se questo rapporto resterà sfavorevole alla difesa, ogni progresso delle capacità offensive dell'IA aumenterà strutturalmente il rischio. Se invece riusciremo a portarlo stabilmente a nostro favore — attraverso rilevamento automatico, controllo indipendente dell'esecuzione e contenimento rapido — l'IA potrà diventare, allo stesso tempo, uno dei più potenti strumenti di attacco e uno dei più potenti strumenti di difesa che la cybersecurity abbia mai avuto.