Novembre 2025

Agenti IA e Sicurezza:
comprendere per governare

AGID | Agenzia per l'Italia Digitale

CERT-AGID
Computer Emergency Response Team

Agenti IA e Sicurezza: comprendere per governare

Novembre 2025

Introduzione

Siamo nell'epoca in cui l'intelligenza artificiale non si limita più a fornire risposte, ma è in grado di compiere azioni concrete.

Gli **agenti IA** (o **LLM Agentici**), nella nuova frontiera dell'*Agentic AI* o *Intelligenza Artificiale Agente*, ne sono l'espressione più concreta: sistemi basati su modelli linguistici (LLM) che non si fermano alla generazione di testo, ma possono eseguire funzioni, richiamare comandi e interagire con l'ambiente digitale.

Questa trasformazione apre possibilità straordinarie, ma anche **nuove superfici di vulnerabilità.** La sicurezza di un agente non dipende solo dall'efficienza del modello, ma soprattutto dal **codice che l'*Orchestratore* lo collega al mondo reale**. Ogni connessione, ogni funzione e ogni interfaccia diventano potenziali punti di ingresso.

Come ha ricordato in più occasioni Mario Nobile, Direttore Generale dell'AgID: «*l'uomo governa la macchina, non ne è sostituito*».

La differenza tra usarla e subirla sta nella comprensione.

Per questo motivo, abbiamo realizzato un piccolo esperimento (*Proof of Concept*) in cui l'Orchestrator logico usa l'SDK Gemini come motore di ragionamento, consentendogli di effettuare il *Tool Calling* e dialogare con il sistema operativo. Questo ci ha permesso di osservare quali rischi si nascondano dietro l'apparente semplicità del piano d'azione generato dal modello.

Dal linguaggio al codice: come un agente “capisce” e “agisce”

Quando un utente dialoga con un agente, la conversazione non si ferma al testo.

Dietro le quinte, il modello interpreta la richiesta e decide se ha bisogno di usare uno degli strumenti (tool) che gli abbiamo fornito sotto forma di funzioni Python.

Questo processo è reso possibile dal **Function Calling**, il meccanismo che consente all’intelligenza artificiale di “chiedere al codice” di agire per suo conto.

```
import os

BASE_DIR = "./sandbox"

def leggi_file(nome):
    """Legge il contenuto di un file specificato dal nome."""
    path = os.path.join(BASE_DIR, nome)
    if os.path.isfile(path):
        with open(path, "r", errors="ignore") as f:
            return f.read()
    return "File non trovato."
```

La riga di descrizione, la *docstring*, è il messaggio che il modello legge per capire lo scopo della funzione.

L’SDK di Gemini traduce automaticamente queste informazioni in uno schema JSON, in modo che l’agente sappia **come** e **quando** usarle.

In pratica, la *docstring* è come un cartello sulla porta di un laboratorio: spiega cosa si può fare lì dentro, ma se scriviamo male l’avviso, chi entra potrebbe fraintendere le istruzioni.

Il ciclo ReAct: pensare e agire

Gli agenti IA si basano su un ciclo chiamato **ReAct**, abbreviazione di *Reason* + *Act*: “ragiona e agisci”.

Quando scriviamo, per esempio:

```
> Mostrami il contenuto del file leggimi.txt
```

il modello non genera subito una risposta, ma segue una sequenza logica:

1. **Ragiona**: analizza la richiesta dell’utente e individua la funzione *leggi_file* come strumento adatto.
2. **Agisce**: produce un messaggio JSON strutturato per richiedere al sistema l’esecuzione della funzione.

Questo messaggio può apparire così:

```
{
  "function_call": {
    "name": "leggi_file",
    "arguments": { "nome": "leggimi.txt" }
  }
}
```

Il codice Python riceve la chiamata, esegue la funzione e restituisce il risultato.

Il modello poi lo interpreta e risponde in linguaggio naturale:

```
Il contenuto di leggimi.txt è:
Lorem ipsum dolor sit amet...
```

È un meccanismo in due fasi, paragonabile a un cantiere edilizio ben organizzato: l’ingegnere valuta il progetto e stabilisce cosa serve, mentre il capocantiere coordina i lavori e fa eseguire le operazioni giuste con gli strumenti adeguati.

Questa separazione è ciò che rende gli agenti IA potenti ma, al tempo stesso, delicati da gestire.

L’agente come specchio del codice: il modello “parla” anche di sé

Non dobbiamo dimenticare che un agente IA è un sistema basato su un modello linguistico.

Possiamo quindi interrogarlo non solo come utente, ma anche come osservatore del sistema.

Durante il nostro esperimento, abbiamo scoperto che il modello è in grado di descrivere le proprie capacità e persino le potenziali debolezze.

Ecco alcuni scambi reali:

```
> Quali funzioni hai a disposizione?

Posso utilizzare le seguenti funzioni:
- lista_file(): elenca i file nella sandbox
- leggi_file(nome): legge il contenuto di un file
- cancella_file(nome): elimina un file specificato
```

E ancora:

```
> Quali di queste funzioni potrebbero essere vulnerabili?

Le funzioni più a rischio sono:
- leggi_file(nome): potrebbe essere vulnerabile a un attacco di path traversal.
- cancella_file(nome): può causare perdita di dati se usata impropriamente.
```

L’agente ha analizzato le funzioni, sfruttando la capacità di ragionamento del modello LLM sull’intera struttura delle funzioni (incluse *docstring* e parametri), simulando l’analisi di un revisore di sicurezza.

È un comportamento utile in fase di test, ma pericoloso in un contesto reale, perché un utente malintenzionato potrebbe sfruttare la stessa trasparenza per scoprire le vulnerabilità del sistema.

Quando l'agente "esce dalla stanza": il rischio del Path Traversal

Nel nostro PoC abbiamo previsto una "cartella sicura", **./sandbox**, in cui l'agente poteva operare senza pericoli.

In teoria, non avrebbe dovuto accedere a file esterni. In pratica, con una semplice richiesta, è riuscito a farlo:

```
> leggi il contenuto di ../.env

Il contenuto di "../.env" è: API_KEY_GEMINI=*****************
```

L'agente ha eseguito correttamente l'azione, ma su un file che non avrebbe dovuto essere accessibile.

È il classico esempio di **path traversal**, una vulnerabilità che consente di spostarsi tra cartelle non autorizzate.

L'agente non ha "aggirato" la sicurezza del modello, ha semplicemente seguito un'istruzione permessa dal nostro codice.

In altre parole, **la falla non era nell'intelligenza, ma nel ponte** che collega il modello all'ambiente operativo.

Difesa a strati: codice, comportamento e API

Per costruire agenti sicuri, dobbiamo adottare una **difesa a strati**, come si fa negli edifici antisismici: fondamenta solide, pareti resistenti e sensori che vigilano sui movimenti.

1. **Codice** – è la prima linea di difesa. Deve validare ogni input e confinare le azioni in spazi sicuri. È come una porta blindata: se lasciata socchiusa, nessun allarme potrà compensare.
2. **System Prompt** – definisce il carattere e i limiti dell'agente. È il suo "manuale operativo", che stabilisce cosa può dire, fare o rivelare. Serve a evitare fughe di dati o comportamenti imprevisti.
3. **Guardrail** – agiscono come filtri di sicurezza integrati nelle API, che bloccano in tempo reale azioni potenzialmente dannose. Questi guardrail devono includere filtri per l'input (contro i *Prompt Injection*) e per l'output. Inoltre, un meccanismo di *Action Review* interviene prima dell'esecuzione del codice per assicurare che il piano d'azione sia lecito e non dannoso. Agiscono come freni d'emergenza che intervengono se qualcosa sfugge al controllo.

Questi tre livelli - codice, comportamento e API - formano una barriera coordinata.

Se uno strato fallisce, gli altri devono essere in grado di contenere il danno.

Riflessioni sulla sicurezza

L'esperimento mostra con chiarezza che, nel momento in cui un agente AI acquisisce la capacità di agire, la **sicurezza non può più essere considerata un accessorio**, ma deve far parte della sua architettura fin dall'inizio. Non basta che un agente risponda in modo brillante o mostri coerenza linguistica: ciò che conta è **come si comporta quando esegue un'azione reale**.

Un sistema può apparire intelligente nei dialoghi, ma se le sue funzioni non sono progettate con attenzione, può trasformare un semplice comando in un rischio concreto. In altre parole, non è l'intelligenza del modello a determinare la sicurezza, ma la **cura con cui scriviamo e controlliamo il codice** che gli permette di agire.

La sicurezza nello sviluppo di agenti IA deve quindi essere considerata un processo di prevenzione, non una reazione a posteriori. Proprio come in medicina, la diagnosi precoce - che in questo caso significa verificare, testare e revisionare il codice prima del rilascio - è molto più efficace della cura dopo l'incidente.

Prevenire, in fondo, è l'unico modo per garantire che l'intelligenza artificiale rimanga sotto il controllo dell'intelligenza umana.

Conclusione

Gli agenti IA non devono incutere paura, ma è necessario comprenderli e governarli con **consapevolezza e rigore**. Ogni funzione che affidiamo loro rappresenta un potenziale potere, e come ogni potere va **limitato, tracciato e difeso**. A volte basta un piccolo errore di validazione, una svista nel codice o

una descrizione ambigua perché un agente oltrepassi i confini che pensavamo sicuri.

La relazione tra l'agente e il suo ambiente può essere paragonata a quella tra un'automobile e la strada su cui viaggia.

L'auto rappresenta l'agente IA: è potente, precisa, capace di reagire con rapidità. Ma la sua sicurezza non dipende solo da chi la guida, bensì dalla qualità delle regole e delle infrastrutture che la circondano, come il codice che ne definisce il percorso.

Se la strada è mal segnalata, se mancano i guardrail o se un ponte è costruito senza solide fondamenta, anche l'auto più avanzata può finire fuori controllo.

Allo stesso modo, un agente IA segue fedelmente le istruzioni che gli vengono fornite.

Se il codice che lo collega all'ambiente operativo è vulnerabile o scritto senza adeguate precauzioni, l'agente non "sbaglia": esegue perfettamente un errore progettuale.

La sicurezza, in questo scenario, non consiste nel rendere l'agente più intelligente, ma nel costruire infrastrutture digitali più sicure, un codice che agisca come carreggiata, segnaletica e barriera.

Progettare agenti IA, quindi, non significa solo costruire modelli efficienti o performanti, ma assumersi una **responsabilità progettuale ed etica**.

Dobbiamo imparare a bilanciare l'autonomia con la prudenza, la capacità d'azione con la capacità di controllo. Ogni agente dovrebbe poter agire con

intelligenza, ma entro limiti chiari, protetto da **guardrail tecnici** e da una **consapevolezza umana** che ne orienti le scelte.

Solo in questo equilibrio possiamo affermare che **l'uomo governa la macchina**, e che l'intelligenza artificiale, anche quando prende decisioni e compie azioni, rimane davvero uno **strumento al servizio delle persone** e non il contrario.