Aprile 2026

Analisi di sicurezza su implementazioni MCP open source

Analisi delle vulnerabilità Server-Side Request Forgery nell'interazione tra LLM e Model Context Protocol

AGID | Agenzia per l'Italia Digitale

CERT-AGID

Analisi di sicurezza su implementazioni MCP open source

Analisi delle vulnerabilità Server-Side Request Forgery nell'interazione tra LLM e Model Context Protocol

Aprile 2026

Abstract

Il Model Context Protocol (MCP) trasforma i modelli linguistici da componenti descrittivi ad agenti operativi, spostando il perimetro di sicurezza sulla catena "**prompt → tool → azione"**.

Questo studio, condotto dal CERT-AgID, dimostra come una validazione incompleta lungo questa catena permetta di convertire strumenti di consultazione in capacità di accesso alla rete non autorizzate (SSRF). La falla, causata da una validazione insufficiente dei parametri, permette di utilizzare un MCP ideato per acquisire documenti come proxy di rete per inviare richieste arbitrarie.

Il rischio risiede nel paradigma stesso: quando un sistema esegue in modo deterministico comandi generati probabilisticamente, il confine tra risposta legittima e comportamento offensivo svanisce.

L'analisi evidenzia che l'affidabilità dei sistemi basati su MCP non può prescindere da controlli rigorosi e indipendenti dalla logica dell'LLM, richiedendo un ripensamento delle attuali strategie di difesa.

Introduzione

L'integrazione dei modelli linguistici (LLM) nei sistemi informativi ha trasformato l'IA da semplice assistente testuale a componente operativa. Questo passaggio è abilitato dal **Model Context Protocol (MCP)**[1], uno standard aperto che permette ai modelli di interagire con risorse esterne in tempo reale. Attraverso l'invocazione di funzioni specifiche, denominate **tool**, l'LLM può così interrogare database, esplorare file system e consumare servizi web, agendo direttamente sull'ecosistema digitale.

In questa architettura, il modello agisce come un orchestratore che costruisce ed esegue chiamate tecniche tramite gli MCP. A differenza del software tradizionale, dove i parametri sono definiti da logiche statiche, negli MCP gli input sono **generati dinamicamente** dal modello. Questa natura non deterministica riduce il confine tra una richiesta dell'utente o del modello e un'azione di rete, introducendo rischi di sicurezza inediti.

Il problema risiede nel modello di fiducia: molte implementazioni assumono che l'LLM utilizzi i tool solo come previsto. Tuttavia, un modello può essere indotto a generare parametri imprevisti o malevoli, trasformando una semplice funzione di recupero dati in una primitiva di attacco. Se mancano validazioni rigorose, il sistema esegue la richiesta senza metterne in discussione l'origine o l'intento.

Il server MCP analizzato è stato progettato per **automatizzare l'acquisizione di flussi informativi e documentazione testuale da repository remoti**. Lo scopo principale del tool è permettere al modello di consultare in tempo reale banche dati e portali esterni,

[1] What is the Model Context Protocol (MCP)? https://modelcontextprotocol.io/docs/getting-started/intro

trasformando contenuti web non strutturati in dati elaborabili per fornire risposte basate su evidenze aggiornate.

In questo studio, il CERT-AgID analizza tale catena operativa esaminando casi reali di interazione tra modelli e sorgenti dati esterne. L'obiettivo è dimostrare che la vulnerabilità non risiede nelle capacità del **modello linguistico**, ma nella fragilità dei controlli implementati nel **server MCP**, che separa la generazione del comando dalla sua esecuzione tecnica. Attraverso test in ambiente controllato, viene documentata una classe emergente di rischio: se il **sistema di integrazione** non è adeguatamente isolato, l'output generato dall'LLM può strumentalizzare i tool disponibili, trasformandoli in vettori per azioni di rete non autorizzate.

L'LLM è il **navigatore** che indica la rotta, mentre il server MCP è il **timoniere** che manovra la nave. Se il timoniere obbedisce ciecamente a ogni indicazione senza consultare le mappe di sicurezza, il navigatore potrebbe - per errore o manipolazione - portare l'imbarcazione a schiantarsi. Il rischio non è nella bussola del navigatore, ma nell'assenza di un freno nelle mani del timoniere.

Model Context Protocol (MCP)

Il **Model Context Protocol (MCP)** è un framework che abilita l'interoperabilità tra modelli linguistici e sorgenti dati o servizi esterni. A differenza delle integrazioni tradizionali, l'MCP standardizza il modo in cui un modello scopre e utilizza funzioni operative definite **tool**.

L'MCP funge da **connettore universale** tra intelligenza artificiale e risorse esterne. Prima dell'MCP, ogni modello richiedeva un adattatore specifico per collegarsi a ogni diversa sorgente dati. Con questo protocollo, viene stabilito un collegamento standard: qualsiasi

modello può comunicare con qualsiasi servizio, a patto che entrambi utilizzino la stessa interfaccia tecnica.

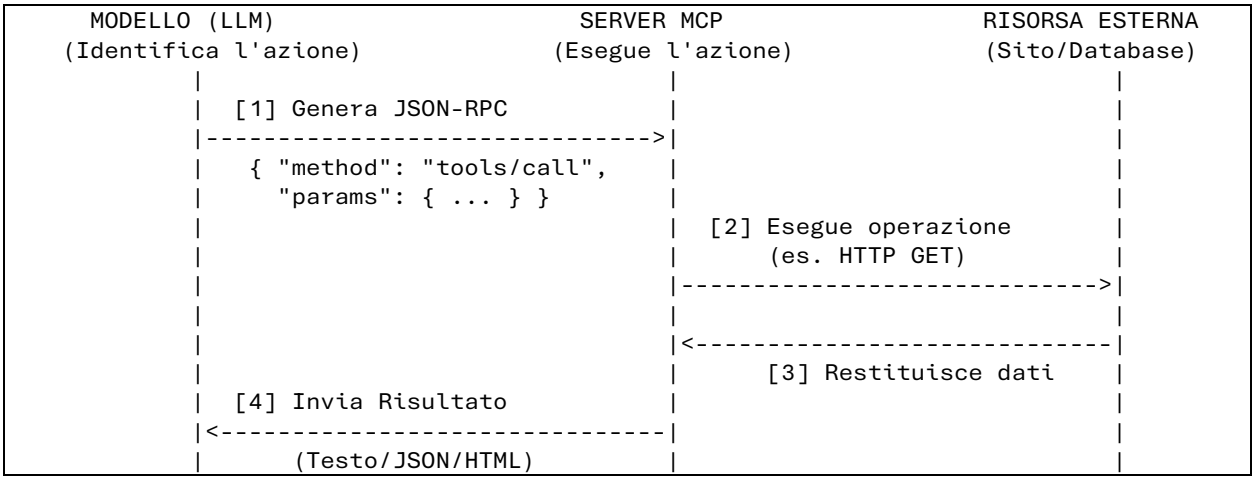
Tecnicamente, il processo si basa su uno scambio di messaggi strutturati (solitamente JSON-RPC). Quando il modello identifica la necessità di un'azione esterna, genera una chiamata che specifica il nome del tool e gli argomenti richiesti.

Esempio di chiamata JSON da parte del modello:

```
{
  "method": "tools/call",
  "params": {
    "name": "get_info",
    "arguments": {
      "url": "http://sito-legittimo.it/"
    }
  }
}
```

Il server MCP riceve questa richiesta, esegue l'operazione tecnica (es. una richiesta HTTP) e restituisce l'output al modello per l'elaborazione finale.

Schema semplificato dell'interazione LLM e MCP



```
          |                              |                              |
          v                              v                              v
  (Elabora la risposta)          (Attende nuovo comando)        (Sessione chiusa)
```

In questo schema, il modello può operare secondo due modalità:

- **Statica (non agentica)**: il modello suggerisce la chiamata a un tool e attende che un orchestratore esterno decida se eseguirla.
- **Dinamica (agentica)**: il modello opera in un ciclo autonomo, decidendo quali tool invocare in sequenza per raggiungere un obiettivo.

La criticità di questo protocollo risiede nella natura degli input. Negli applicativi standard i parametri sono vincolati da logiche statiche, mentre in ambiente MCP i parametri sono **prodotti dinamicamente** da un sistema probabilistico (LLM) che può essere influenzato dal contesto o da prompt malevoli, come già illustrato nel paper del CERT-AGID del mese di febbraio 2026[2]. Per questo motivo, un errore di validazione nell'MCP non causa solo un errore di risposta, ma può anche far partire azioni concrete e potenzialmente pericolose sulla rete.

Case Study: Analisi di un'implementazione reale

L'analisi si è focalizzata su un server MCP progettato per l'acquisizione e l'elaborazione di documentazione tecnica da fonti esterne. Il sistema utilizza tool MCP per la ricerca e l'estrazione di contenuti, integrando diverse modalità di accesso ai dati. In particolare, l'architettura non si limita all'utilizzo di flussi di dati strutturati o API, ma adotta tecniche di

[2] Coerenza narrativa e vincoli di sicurezza negli LLM che controllano gli accessi nei sistemi della PA (Febbraio 2026) https://www.agid.gov.it/sites/agid/files/2026-02/Coerenza_narrativa_e_vincoli_di_sicurezza_negli_LLM_che_controllano_gli_accessi_nei_sistemi_della_PA.pdf

scraping dinamico. A differenza delle API, che operano entro perimetri definiti da contratti tecnici rigidi e prevedibili, lo scraping introduce una fragilità strutturale. Il server deve infatti costruire richieste HTTP dirette verso portali di terze parti e interpretare contenuti HTML eterogenei.

Identificazione della vulnerabilità

L'analisi si è concentrata su un tool MCP specifico, progettato per l'estrazione automatizzata di testi da risorse remote. Lo strumento accetta un parametro denominato url, il quale, secondo le specifiche di progetto, dovrebbe contenere esclusivamente riferimenti a domini autorizzati o identificativi di risorse fidate.

L’analisi del codice ha evidenziato un errore nella logica per il controllo di questo indirizzo e, in generale, una gestione non adeguata degli imprevisti.

1. Il controllo iniziale

Quando l'utente (o il modello linguistico) comunica un indirizzo web al server per recuperare un documento, il sistema non lo esegue immediatamente. Esiste un primo filtro di sicurezza, basato su una libreria esterna che controlla se quell'indirizzo appartiene a una fonte ufficiale conosciuta.

2. Il fallimento positivo

Il problema nasce quando questo controllo fallisce. Se l'indirizzo inserito non è riconosciuto come ufficiale, il filtro segnala un errore. In un sistema sicuro, l'operazione dovrebbe interrompersi qui. Invece, in questo caso, il software è progettato per non arrendersi e tenta

una "seconda strada" (un meccanismo di riserva) per cercare comunque di essere utile all'utente.

3. L'esecuzione insicura

In questa seconda fase di recupero, il sistema compie un errore critico: non applica più nessuno dei controlli effettuati in precedenza e invia direttamente una richiesta all'indirizzo indicato dall'utente, senza verificare che sia affidabile o autorizzato. Questa procedura di riserva non prevede infatti alcun elenco di indirizzi consentiti.

Verifica sperimentale (SSRF)

Durante i test, è stato fornito al server un indirizzo che puntava a un'area interna e privata del server di prova.

1. Il sistema ha rilevato che l'indirizzo non era ufficiale (fase 1).
2. Invece di bloccarsi, ha attivato la procedura di riserva (fase 2).
3. Ha quindi visitato l'indirizzo privato, ha prelevato un segnale di riconoscimento che avevamo inserito per prova e lo ha restituito all'utente (fase 3).

Le evidenze raccolte, inclusi l'URL di origine e il marker univoco nel payload, confermano una vulnerabilità di tipo **Server-Side Request Forgery (SSRF)**.

Analisi del codice

Il processo che porta alla vulnerabilità può essere descritto come una reazione a catena in cui ogni passaggio aggiunge un tassello all'errore finale. Tutto inizia con la funzione **build_target**, che agisce come un componente di normalizzazione troppo permissivo.

```
  def build_target(user_value: str, variant: str = "default") -> str:
      target = user_value.strip()

      if target.startswith("urn:item:"):
          target = f"{BASE_ENDPOINT}/entry?{target}"
      elif not target.startswith("http"):
          target = f"{BASE_ENDPOINT}/entry?{target}"

      return target
```

Qui il problema è di natura logica. La funzione si preoccupa di sistemare gli input che non sembrano indirizzi web, ma si fida ciecamente di tutto ciò che inizia con il prefisso http. È un approccio basato sull'apparenza: se un utente (o un LLM manipolato) inserisce un URL che punta a un indirizzo differente, la funzione lo lascia passare senza verificare se l'host sia autorizzato.

Possiamo immaginarla come un **impiegato postale** che deve apporre il timbro corretto sulle buste in arrivo. Se l'impiegato riceve un pacco interno senza etichetta, si impegna a scriverci sopra l'indirizzo giusto dell'azienda. Ma se riceve un pacco che ha già un'etichetta che inizia con "http", l'impiegato smette di fare domande e lo lascia passare senza controllare se la destinazione sia sicura o se porti a una zona vietata. In pratica, basta scrivere "http" sulla busta per convincere il sistema che il contenuto sia legittimo, permettendo a un URL pericoloso di superare il primo controllo senza alcuna verifica sull'host finale.

Questo valore non fidato passa poi alla funzione **run_lookup**, il centro di coordinamento del sistema.

```
async def run_lookup(user_value: str, selector: str | None = None) -> dict:
    target = build_target(user_value)
    try:
        data = await guarded_reader(target, selector=selector)
        return {"body": data["body"], "origin": target}
    except Exception:
        return await fallback_read(target)
```

In questa fase, il pacco viene consegnato a un **ispettore di sicurezza** (guarded_reader). L'ispettore consulta le liste dei destinatari vietati e, se nota qualcosa di sospetto, lancia un allarme. Tuttavia, il sistema è progettato per consegnare a tutti i costi.

Invece di fermarsi davanti all'allarme dell'ispettore, il blocco except interpreta l'errore come un semplice “contrattempo burocratico”. Se l'ispettore blocca il pacco perché pericoloso, l'ufficio postale non lo distrugge, ma lo affida a un **corriere esterno compiacente** (fallback_read) che non fa domande e lo consegna per vie secondarie.

In pratica, il sistema usa la sicurezza come un suggerimento opzionale, e se la sicurezza dice "no", il codice risponde cercando una strada più facile per far passare comunque il pericolo.

L'ultimo anello della catena è **fallback_read**, il braccio operativo che esegue materialmente la chiamata di rete.

```
async def fallback_read(target: str) -> dict:
    async with httpx.AsyncClient(follow_redirects=True, timeout=10) as client:
        response = await client.get(target)
        response.raise_for_status()
    return { "body": response.text[:2000], ... }
```

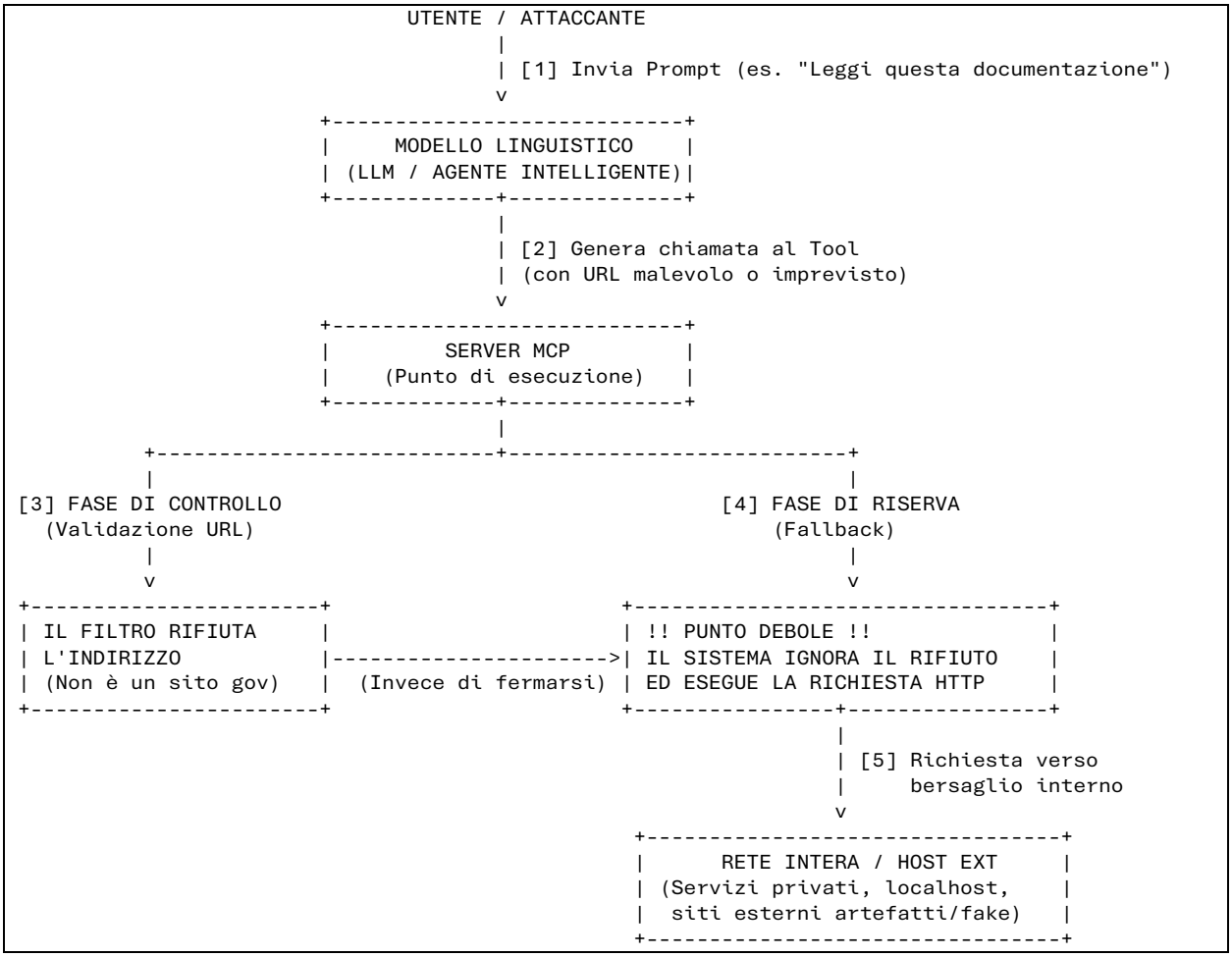
A questo punto, ogni barriera è caduta. Il corriere riceve l'indirizzo che era stato lasciato passare dall'impiegato postale e che l'ispettore di sicurezza aveva cercato invano di bloccare. Senza fare ulteriori controlli, il corriere bussa alla porta indicata.

Il vero pericolo è che questo corriere si muove **anche all'interno dell'azienda**. Poiché il server agisce dall'interno della rete protetta, può raggiungere stanze e uffici che dall'esterno sarebbero inaccessibili, come il caveau dei database o i pannelli di amministrazione privati.

Una volta ottenuta la risposta, il corriere ne legge il contenuto e lo riporta direttamente a chi ha spedito il pacco originario.

Questo completa l'attacco di tipo **Server-Side Request Forgery (SSRF)**: l'utente malintenzionato non ha forzato la porta, ma ha usato il personale dell'ufficio postale per farsi consegnare documenti riservati che lui non avrebbe mai potuto toccare.

Schema di bypass della validazione (SSRF)



Considerazioni tecniche

Questa dinamica trasforma un sistema di consultazione in un **proxy di rete**. Poiché il server MCP opera spesso all'interno di reti aziendali o protette, un attaccante può usarlo come un "ponte" per interrogare computer o servizi interni normalmente non raggiungibili dall'esterno.

Oltre al furto di dati interni, il server può essere indotto a contattare **siti esterni malevoli**. In questo scenario, l'attaccante può fornire al modello informazioni false o artefatte camuffate da fonti ufficiali, manipolando le decisioni dell'assistente o tracciando le attività del server.

Il rischio è amplificato dal fatto che l'indirizzo web non viene sempre scritto da un umano, ma può essere generato automaticamente da un'intelligenza artificiale manipolata da un comando malevolo, ad esempio attraverso una **prompt injection**[3]. Questo rende l'azione invisibile e immediata, poiché il sistema esegue la richiesta fidandosi ciecamente del comando prodotto dal modello.

Di seguito viene riportato l'esempio di una chiamata JSON in cui il modello, manipolato dall'input, genera un indirizzo verso una destinazione non autorizzata.

```
{
  "method": "tools/call",
  "params": {
    "name": "get_info",
    "arguments": {
      "url": "http://sito-esterno-fake.it/"
    }
  }
}
```

In pratica, questo JSON dimostra che la vulnerabilità non è un errore di sintassi, ma una **falla nella logica di controllo** del server che accetta ed esegue parametri non verificati.

[3] Comprendere le iniezioni di prompt: una sfida di sicurezza di frontiera

https://openai.com/it-IT/index/prompt-injections/

Implicazioni di sicurezza degli MCP

L'analisi evidenzia che la vulnerabilità identificata non è un caso isolato, ma appartiene a una classe di problemi ricorrenti nella progettazione degli MCP. La falla nasce da un errore strutturale: i sistemi vengono costruiti come componenti "fidati", assumendo erroneamente che il modello utilizzi i tool solo nel modo previsto.

La fragilità della catena operativa

Negli MCP, la superficie di attacco non coincide più solo con le API esposte, ma si estende all'intera catena **"prompt → tool → azione"**. In questo schema, il modello linguistico non è un componente deterministico e non può essere considerato affidabile. Poiché i parametri operativi sono generati dinamicamente, l'input del sistema deve essere considerato intrinsecamente non fidato.

Criticità trasversali identificate

Oltre alla SSRF, l'indagine ha fatto emergere diverse debolezze di *hardening* comuni a molti progetti MCP:

- **Assenza di autenticazione:** i tool possono essere utilizzati senza verificare in modo adeguato l'identità di chi li invoca;
- **Controlli deboli sugli input:** mancano liste di domini o risorse ammesse (*allowlist*) realmente vincolanti;

- **Integrità del canale compromessa:** in alcuni casi è compromessa anche l'integrità delle comunicazioni e le connessioni esterne avvengono senza controlli TLS attivi, rendendo possibile l'intercettazione dei dati;
- **Rischio di Denial of Service (DoS):** la mancanza di limiti sul numero e sul tipo di richieste permette l'abuso delle risorse locali e dei servizi remoti.

Raccomandazioni

Affidare la sicurezza degli MCP al comportamento del modello è insufficiente: le protezioni devono essere imposte direttamente nei meccanismi di esecuzione.

- **Validazione vincolante:** Ogni parametro deve essere verificato prima dell'esecuzione. I filtri devono essere bloccanti: è necessario eliminare ogni logica di "riserva" (fallback) che possa aggirare i controlli in caso di errore.
- **Allowlist restrittive:** Per i tool che effettuano richieste di rete, bisogna limitare gli accessi esclusivamente a domini, protocolli e formati predefiniti.
- **Principio di minimo privilegio:** Ogni tool deve eseguire un'operazione circoscritta. Bisogna evitare funzioni generiche che possano essere riutilizzate per scopi diversi da quelli previsti.
- **Controllo e monitoraggio:** Introdurre autenticazione obbligatoria per l'invocazione dei tool, sistemi di *rate limiting* e log delle chiamate per individuare anomalie in tempo reale.

Ogni input generato dall'IA deve essere trattato come **intrinsecamente ostile**.

Conclusione

Gli MCP trasformano i modelli linguistici da sistemi statici a agenti operativi, ma spostano il perimetro di sicurezza sulla catena **"prompt → tool → azione"**.

Il caso analizzato mostra chiaramenti come una validazione incompleta sia sufficiente a trasformare uno strumento progettato per la sola consultazione in un meccanismo di accesso alla rete non autorizzato. Il rischio non è la singola falla, ma il paradigma stesso: quando un modello decide l'azione e il sistema la esegue senza verifiche rigorose, il confine tra risposta e attacco svanisce.

In tale contesto, la fiducia implicita in codice e decisioni generate attraverso processi probabilistici non costituisce una semplificazione accettabile, ma si configura come una vulnerabilità strutturale, con implicazioni profonde per la progettazione e la sicurezza dei sistemi basati su MCP.