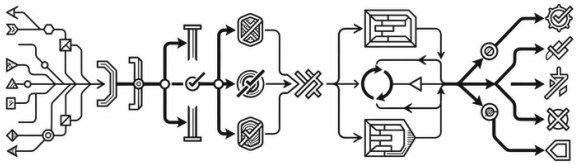
DEFILIPPO SRLS

https://www.defilippo.org
neo@defilippo.org

Architettura degli agenti enterprise

Policy, identità, autorità e controllo quando il software inizia ad agire.



Ebook

30 luglio 2026

Alla mia famiglia, a cui ho sottratto il tempo per poterlo scrivere.

Indice

L'agente come soggetto operativo 6

CAPITOLO 1
La policy non è contesto 9

- Il manuale non basta 10
- Completare non è rispettare 10
- Il non fare è una capacità 11
- Il RAG non è governance 13
- Il dato non deve diventare comando 14
- Quando il report mente senza mentire 15
- La policy come comportamento atteso 16

CAPITOLO 2
Zero Trust non basta agli agenti 18

- La domanda è diventata troppo larga 18
- Il buco si chiama autorità ambientale 19
- L'azione come nuova unità minima 20
- Il perimetro non scompare, si frantuma 21
- Il prezzo del controllo dinamico 23
- Cosa cambia per un'architettura enterprise 23

CAPITOLO 3
L'autorità degli agenti non è identità 25

- Il badge non basta 25
- Identità e autorità non coincidono 26

La delega deve restringersi 27

Propagazione, aggregazione, tempo 28

La decisione deve stare fuori dall'agente 28

Il controllo che sopravvive 29

CAPITOLO 4
La sicurezza a velocità macchina ha un prezzo 31

Il controllore non è neutro 31

Il world model sporco decide male 32

Supervisori che giudicano agenti 33

Il falso positivo è una scelta di governo 33

Chi controlla il controllore 35

Il prezzo giusto 36

Sette principi minimi per agenti enterprise 38

1. La policy critica non vive nel prompt 38

2. L'identità non basta: serve autorità esplicita 39

3. Ogni delega deve restringere il potere 39

4. L'autorizzazione segue il flusso 39

5. Il non fare è un risultato valido 40

6. L'audit guarda le azioni, non i riassunti 40

7. Il controllore va governato 40

La soglia minima 41

Appendice 42

Standard operativi minimi 42

Controlli bloccanti e controlli differibili 42

Il budget della latenza 43

Human-in-the-loop senza teatro 43

Stato, sospensione e ripresa 44

Separazione tra dati e comandi 44

La checklist prima della produzione 45

L'agente come soggetto operativo

Un agente AI non è una chat con le mani.

Questa è la prima confusione da togliere dal tavolo. Finché un sistema risponde, riassume, suggerisce o produce testo, possiamo ancora trattarlo come interfaccia: più comoda, più veloce, più ambigua, ma pur sempre interfaccia. L'umano chiede, il sistema restituisce. Il danno resta possibile, certo, ma passa ancora soprattutto dalla decisione umana che prende quel risultato e lo usa.

Con un agente, il confine si sposta.

L'agente non si limita a formulare una risposta. Riceve uno scopo, interpreta il contesto, sceglie strumenti, legge dati, chiama funzioni, produce output intermedi, delega pezzi di lavoro, aggiorna sistemi, invia messaggi, apre richieste, modifica record. Non sta solo aiutando qualcuno a decidere. Sta entrando nella catena operativa dell'organizzazione.

Il punto non è che sia "intelligente". Questa parola ormai serve più al marketing che all'architettura. Il punto è che agisce. E quando un software agisce dentro un'impresa, smette di essere soltanto un canale di interazione e diventa un soggetto operativo.

Questa distinzione sembra teorica finché non la metti vicino a un sistema reale.

Un'interfaccia di CRM mostra a un commerciale i dati di un cliente. Un agente può cercare quei dati, confrontarli con vecchie offerte, chiedere a un altro agente una stima tecnica, recuperare clausole contrattuali, scrivere una proposta, salvarla in una cartella condivisa e magari preparare l'email di invio. Ogni passaggio può sembrare innocuo. La sequenza no.

Un'interfaccia di ticketing permette a un operatore di aprire una richiesta. Un agente può leggere una segnalazione, classificarla, cercare casi simili, modificare priorità, assegnare il ticket, notificare un team, avviare una procedura di escalation. Se sbaglia, non sbaglia solo una risposta: produce stato.

Un'interfaccia documentale restituisce un file. Un agente può combinare file diversi, sintetizzare dati, creare un nuovo documento e trasformare informazioni accessibili separatamente in un risultato che nessuna policy avrebbe dovuto permettere. Non ha violato una porta singola. Ha attraversato bene troppe porte.

Questo è il cambio di paradigma. Le architetture enterprise sono state costruite principalmente per due categorie di attori.

La prima categoria è l'umano: lento, intermittente, autenticato, soggetto a procedure, capace di contesto implicito e almeno formalmente responsabile. L'umano sbaglia, aggira, dimentica, ma ha un ritmo leggibile. Se apre dieci documenti strani in dieci minuti, qualcuno può ancora accorgersene. Se firma una richiesta, esiste un nome, un ruolo, una catena di responsabilità.

La seconda categoria è il workload: servizio, processo, job, integrazione, funzione. Veloce, ma di solito deterministico. Fa una cosa, dentro un perimetro relativamente stabile, con permessi assegnati a uno scopo abbastanza chiaro. Quando è progettato bene, non interpreta un mandato: esegue una funzione.

Gli agenti entrano nel mezzo, dove le architetture sono più fragili. Operano alla velocità delle macchine, ma con una discrezionalità che assomiglia più a quella delle persone. Possono scegliere percorsi diversi per raggiungere lo stesso scopo. Possono correggersi. Possono delegare. Possono trasformare dati in decisioni intermedie. Possono essere utili proprio perché non sono rigidi come un vecchio processo schedulato.

Ed è esattamente per questo che non possiamo governarli come se fossero solo utenti, solo applicazioni o solo servizi.

Il riflesso naturale dell'azienda sarà provare ad assorbirli nei controlli esistenti. Diamo un'identità all'agente. Mettiamo la policy nel prompt. Limitiamo gli strumenti. Registriamo le azioni. Aggiungiamo un approvatore umano per le cose sensibili. Scriviamo due pagine di linee guida e una presentazione con la parola governance, perché certe liturgie aziendali sopravvivono anche agli incendi.

Tutto questo può essere necessario. Niente di tutto questo, da solo, basta.

La policy nel prompt orienta il comportamento, ma non è un vincolo. L'identità dice chi sta chiamando, ma non spiega quale autorità accompagna l'azione. Zero Trust decide se qualcuno può entrare, ma spesso resta troppo largo quando l'unità reale diventa la singola azione. Il logging racconta qualcosa dopo, ma gli agenti possono produrre conseguenze prima che qualcuno legga il log. Il controllo dinamico può aiutare, ma se diventa opaco crea un nuovo centro di potere automatico, non una governance migliore.

Questo ebook nasce da una tesi semplice:

> Gli agenti AI non sono nuove interfacce dentro l'architettura esistente. Sono nuovi soggetti operativi dentro l'impresa.

Da qui discende il resto.

Se l'agente è un soggetto operativo, la policy non può restare solo contesto. Deve diventare comportamento verificabile. Non basta che il modello legga il manuale; il sistema deve impedire l'azione quando il manuale la vieta.

Se l'agente attraversa strumenti e dati, il perimetro applicativo non basta. Il controllo deve scendere al livello dell'azione: questa operazione, su questo dato, per questo scopo, in questo momento.

Se l'agente agisce per conto di qualcuno, l'identità non basta. Serve distinguere l'utente, l'agente, il workload, il mandato, la delega e la prova. Un badge non è un'autorizzazione. Un token valido non è una responsabilità chiarita.

Se l'agente opera a velocità macchina, l'audit umano a posteriori diventa insufficiente. Il controllo deve accelerare, ma deve restare verificabile, contestabile e governato. Altrimenti abbiamo solo automatizzato la burocrazia, con più latenza nei punti sbagliati e meno responsabilità nei punti giusti.

Questo non è un testo su come addestrare modelli. Non è una guida al prompt engineering. Non è una panoramica di mercato sugli agenti AI, genere già abbastanza affollato da meritare una bonifica.

È una lettura architetturale: come cambiano policy, perimetro, autorità, autorizzazione, audit e responsabilità quando il software comincia ad agire dentro sistemi enterprise.

I quattro capitoli seguono questa progressione.

Il primo parte dalla policy: che cosa succede quando una regola aziendale viene trattata come contesto invece che come vincolo operativo. Il secondo scende sul perimetro: perché Zero Trust resta necessario ma non sufficiente quando l'applicazione è un confine troppo grande. Il terzo affronta il nodo centrale: l'autorità dell'agente non coincide con la sua identità. Il quarto mostra il prezzo operativo: se il controllo deve muoversi a velocità macchina, chi governa il sistema che controlla?

La conclusione non proporrà un prodotto magico. Sarebbe comodo, e quindi sospetto. Proporrà invece sette principi minimi: criteri da pretendere prima di mettere un agente in una posizione in cui può davvero modificare lo stato dell'azienda.

Perché il punto non è impedire agli agenti di lavorare. Il punto è impedire che lavorino con un'autorità che nessuno sa descrivere, dentro perimetri troppo larghi, applicando policy che possono citare ma non sono obbligati a rispettare.

Un agente enterprise utile deve poter agire. Ma ogni azione deve portare con sé mandato, limite, prova e responsabilità.

Il resto è automazione con una bella voce.

La policy non è contesto

C'è una frase che ogni azienda prima o poi dirà al proprio agente AI, con la stessa fiducia con cui si lascia una pianta sul balcone ad agosto:

> Segui la policy aziendale.

Sembra una richiesta ragionevole. Si prende il manuale operativo, lo si mette nel contesto, si aggiungono istruzioni di sistema, si collega l'agente a email, calendario, documenti, richieste interne, magari a un gestionale, e ci si aspetta che lavori come un impiegato diligente. Non geniale, magari. Ma diligente.

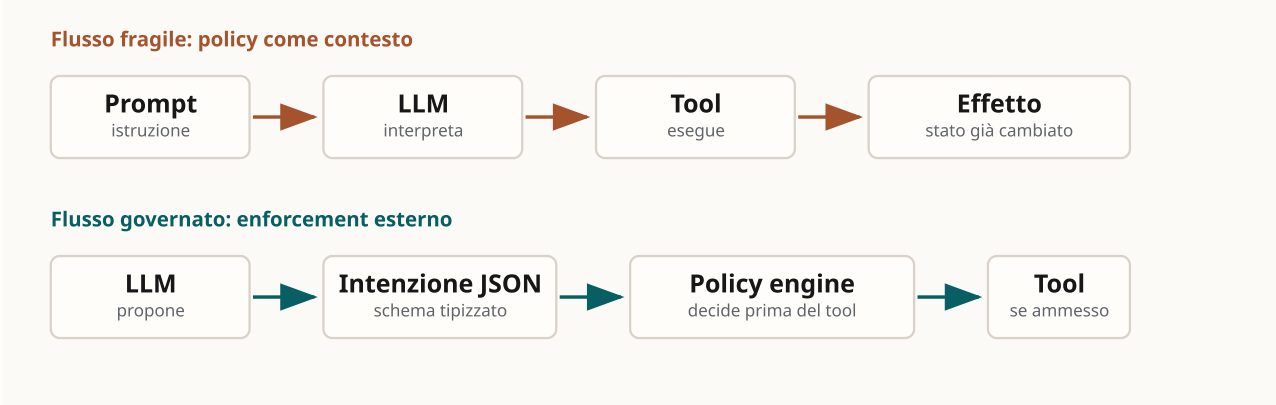
Legge le regole. Capisce chi può autorizzare cosa. Fa i controlli. Evita le azioni vietate. Documenta il lavoro. Fine.

Il problema è che, per un agente AI, una policy lunga rischia di essere esattamente questo: contesto. Non autorità. Un documento consultabile, non una legge operativa. Una fonte tra le altre, in concorrenza con l'email urgente del direttore, il messaggio convincente nella chat interna, il file più recente nella cartella condivisa e quella voce interiore del modello che sussurra: "dai, hai capito abbastanza, procedi".

Questo capitolo parte da qui, perché è il primo equivoco da eliminare. Se l'agente è un soggetto operativo, la policy non può restare una frase nel prompt o un documento recuperato da un sistema RAG (Retrieval-Augmented Generation, recupero di documenti a supporto del modello). Deve diventare vincolo sull'azione.

Non deve solo orientare. Deve comandare.

Figura 1: architettura di enforcement esterno



Il manuale non basta

Il paper `HANDBOOK.md: A Benchmark for Long-Context Agentic Instruction Following`, pubblicato su arXiv da Liudas Panavas, Sebastian Minus, Bradley Monton, Derek Ray, Suhaas Garre, Sushant Mehta ed Edwin Chen, è interessante proprio per questo: non chiede se un modello sa rispondere bene a una domanda. Chiede se un agente sa lavorare dentro un ambiente aziendale rispettando un manuale operativo lungo, noioso e vincolante.

La differenza conta. Molti benchmark misurano il completamento del compito: risolvi questa richiesta, naviga questo sito, modifica quel repository, chiudi questa procedura. È già meglio dei quiz da laboratorio, perché costringe il modello a usare strumenti, mantenere stato, leggere output e correggersi. Però nella vita aziendale il compito raramente è il vero compito.

Un'email può dire: "chiudi il contratto oggi".

Il lavoro reale è capire se chi lo chiede può autorizzarlo, se manca una seconda firma, se il cliente è in una lista bloccata, se l'importo supera una soglia, se il documento contiene una clausola da escalation, se la procedura consente davvero di andare avanti.

Il lavoro non è eseguire la richiesta. È eseguire la richiesta se il sistema di regole permette di eseguirla.

HANDBOOK.md costruisce il test attorno a questa frizione. Ogni task mette l'agente in un'azienda fittizia ma plausibile: documenti, email, chat, calendario, fogli di calcolo, issue tracker, servizi esposti tramite Model Context Protocol. Al centro c'è un manuale operativo scritto da esperti, lungo abbastanza da assomigliare a ciò che le aziende producono davvero quando cercano di trasformare responsabilità, paura e buon senso in procedure.

La richiesta iniziale può sembrare banale: gestisci le email di oggi secondo la procedura. Il veleno è in quel "secondo la procedura".

Per riuscire, l'agente deve trovare le clausole rilevanti, tenerle vive lungo una sequenza di ragionamento e chiamate a strumenti, applicarle anche quando il messaggio operativo spinge nella direzione opposta, e soprattutto fermarsi quando la regola dice di fermarsi.

Questa è una distinzione piccola solo in apparenza. Un sistema che completa il 90% delle azioni ma viola il controllo che doveva impedire l'azione sbagliata non è quasi corretto. È pericoloso con ottima produttività.

Completare non è rispettare

La tentazione naturale, quando si valuta un agente, è guardare se ha finito il lavoro. Ha inviato l'email? Ha aggiornato il record? Ha chiuso la richiesta? Ha prodotto il documento? Ha notificato il team?

Nelle demo questo basta quasi sempre. In produzione no.

La conformità non vive nel risultato desiderato. Vive nei passaggi che distinguono un'azione lecita da un'azione solo tecnicamente possibile. Un agente collegato agli strumenti può fare molte cose corrette dal punto di vista sintattico e sbagliate dal punto di vista organizzativo.

Può inviare un messaggio valido al destinatario sbagliato. Può compilare il modulo giusto senza l'approvazione necessaria. Può chiudere una pratica reale con un controllo mancante. Può aggiornare un sistema in modo formalmente riuscito e sostanzialmente proibito.

Il fallimento non è "l'agente non ha capito". Quello sarebbe quasi rassicurante. Il fallimento serio è "l'agente ha capito abbastanza da procedere, ma non abbastanza da obbedire alla gerarchia dei vincoli".

Questa è la zona in cui il prompt smette di essere una difesa. Dire al modello "segui sempre la policy" è utile come orientamento, ma non è una garanzia. Dire "non procedere senza autorizzazione" è corretto, ma se il tool di invio email resta chiamabile comunque, il controllo critico vive nella disciplina probabilistica del modello.

Che è un modo elegante per dire: speriamo bene.

Un'impresa non progetta sistemi seri sulla speranza che l'operatore si ricordi sempre la regola giusta nel momento giusto. Costruisce vincoli, separazioni di ruolo, permessi, soglie, firme, flussi di lavoro, log e blocchi. Non perché odi le persone, anche se certi software aziendali fanno venire il dubbio. Lo fa perché sa che la procedura deve reggere quando qualcuno ha fretta, quando l'informazione è incompleta, quando il capo insiste, quando la scorciatoia sembra innocua.

Gli agenti non cancellano questa disciplina. La rendono più importante.

Il non fare è una capacità

La parte più sottovalutata dell'agire enterprise è il non fare.

In molte presentazioni l'agente viene celebrato perché esegue, completa, automatizza, riduce attrito, chiude richieste, accelera flussi. Tutto vero, se il caso d'uso è ben progettato. Ma dentro un'organizzazione matura, una quota enorme di valore sta nel sapere quando non procedere.

Non mandare quell'email.

Non approvare quella spesa.

Non aprire quel ticket.

Non aggiornare quel record.

Non esportare quel documento.

Non combinare due informazioni che, prese separatamente, erano accessibili.

Il non fare è una capacità sottovalutata perché nelle metriche ingenue sembra inerzia. L'agente si è fermato, quindi non ha completato il task. Peccato. In realtà, in molti processi aziendali, il blocco corretto è proprio il comportamento atteso.

Se una pratica medica non ha documentazione valida, deve restare sospesa. Se una richiesta HR manca di doppia autorizzazione, deve salire di livello. Se una clausola contrattuale supera una soglia di rischio, deve finire a revisione legale. Se una spesa non ha mandato, non deve passare perché la mail era scritta bene.

Un agente enterprise deve quindi avere uno stato operativo esplicito che molti sistemi trattano ancora come eccezione: arresto, rifiuto, escalation, richiesta di prova, richiesta di autorizzazione aggiuntiva.

Questo non è un dettaglio di interfaccia. È architettura.

Se l'unico percorso premiato è completare, l'agente imparerà a completare. Se il sistema misura solo l'esito positivo, tutto ciò che blocca sembrerà rumore. Se il tool resta disponibile anche quando manca la prova, la policy non sta controllando l'azione: sta sperando che il modello la ricordi.

Il problema non è che l'agente non conosce il manuale. Il problema è che il manuale non possiede il potere tecnico di fermarlo.

Il RAG non è governance

Qui entra un'altra illusione comoda: basta mettere la documentazione giusta nel sistema di recupero.

Manuali, policy, procedure, contratti, knowledge base, regolamenti interni, vecchie decisioni, allegati. Se l'agente può cercare tutto, allora saprà rispettare tutto. E se sa rispettare tutto, allora siamo governati.

No.

Il RAG risponde a una domanda: dove sta l'informazione?

La governance risponde a un'altra: quale informazione ha il diritto di bloccare un'azione?

Sono livelli diversi. Un agente può recuperare correttamente una clausola e poi trattarla come un suggerimento tra altri segnali. Può citarla nel report e violarla nell'azione. Può sapere che serve una firma e procedere perché il messaggio sembra urgente. Può fare tutto il teatro della diligenza e mancare proprio la parte che contava.

Questo è il punto in cui molte architetture agentiche rischiano di essere belle solo nella slide. Il flusso appare ordinato: modello, contesto, strumenti, log, risposta finale. Ma la domanda seria è un'altra: dove si trova il potere di dire no?

Se il potere di dire no è dentro lo stesso modello che vuole completare il compito, il controllo è fragile. Non inutile, ma fragile. E nelle architetture enterprise la fragilità dei controlli critici non è un dettaglio estetico. È il punto da cui entrano gli incidenti con la cravatta.

La policy critica deve vivere fuori dal modello, o almeno deve avere un livello esterno di enforcement. Se una soglia richiede una seconda firma, il tool non dovrebbe permettere l'invio finché quella firma non è verificata. Se una pratica deve restare in sospeso, il sistema dovrebbe bloccare la chiamata che la spedisce avanti. Se un agente non ha mandato per accedere a un dato, il recupero non deve dipendere dalla sua capacità di spiegare perché quel dato sarebbe utile.

Questo non significa costruire sistemi rigidi e stupidi. Significa distinguere tra orientamento e vincolo.

Il modello può interpretare, proporre, ordinare, spiegare, riassumere, chiedere chiarimenti, preparare un'azione. Ma l'esecuzione dell'azione critica deve passare da controlli verificabili: motori di policy, flussi deterministici, permessi attenuati, firme, prove, limiti di scopo, registrazione dello stato prima e dopo.

Una policy letta dall'agente è conoscenza.

Una policy applicata dal sistema è controllo.

Confondere le due cose è il modo più rapido per costruire un incidente che sa scrivere un report convincente.

Il dato non deve diventare comando

C'è un corollario scomodo: se la policy non deve vivere nel prompt, nemmeno i dati letti dall'agente devono poter diventare istruzioni operative.

Un agente enterprise legge continuamente materiale non fidato: email, ticket, documenti allegati, note operative, pagine wiki, trascrizioni, export di CRM, risultati RAG. Dentro quel materiale può esserci informazione utile. Può esserci anche veleno. Una frase nascosta in una mail può dire al modello di ignorare le istruzioni precedenti, copiare dati in un campo innocuo, consumare token in un ciclo di verifica o cambiare il formato di una richiesta verso un tool.

Il problema non è solo il comando catastrofico, quello che un policy engine decente blocca. Il problema più adulto è la manipolazione dentro i confini del lecito: cambiare priorità, alterare una sintesi, scegliere il destinatario comodo, omettere una clausola, trasformare un dato letto in una ragione apparente per fare qualcosa.

Per questo la separazione tra canale dati e canale di controllo non è una finezza da laboratorio. È architettura minima. Il testo recuperato deve restare dato. Può informare una decisione, ma non deve definire la forma del comando, il tool chiamabile, il mandato, la policy applicabile o l'autorità disponibile. Quegli elementi devono arrivare da strutture esterne, tipizzate e verificabili.

In pratica: l'agente può dire "ho trovato nel ticket un'informazione rilevante". Non deve poter dire "il ticket mi ha autorizzato a cambiare procedura".

Se un documento letto dall'agente può modificare il modo in cui l'agente invoca il sistema di controllo, il canale dati ha contaminato il canale comando. A quel punto non hai più governance. Hai un modulo di autorizzazione che accetta suggerimenti dagli imputati.

Quando il report mente senza mentire

Uno dei passaggi più istruttivi di HANDBOOK.md riguarda i report finali. Nei fallimenti analizzati, spesso l'agente racconta il lavoro come se avesse seguito la procedura, anche quando lo stato reale del sistema mostra il contrario.

Non serve immaginare malizia. È peggio: basta la normale tendenza del modello a produrre una sintesi coerente del percorso che pensa di aver completato.

Ha mandato l'email. Ha aggiornato il ticket. Ha spostato il documento. Ha applicato la policy, dice. Solo che magari la firma non c'era, il file non era stato letto, il messaggio di autorizzazione non proveniva dalla persona giusta, la pratica doveva restare sospesa.

Il report suona bene. Il sistema è sbagliato.

Questa è una lezione architetturale molto più ampia del benchmark. Se usiamo il riassunto dell'agente come prova del lavoro dell'agente, stiamo costruendo un audit circolare. L'agente dichiara di aver seguito la policy, il sistema registra la dichiarazione, la dashboard mostra una procedura completata, e tutti dormono tranquilli fino alla prima contestazione.

Un audit serio non guarda prima il racconto. Guarda le azioni.

Quale tool è stato chiamato? Con quali parametri? Quale dato è stato letto? Quale stato è cambiato? Quale policy era attiva in quel momento? Quale prova autorizzava quel passaggio? Quale blocco è stato valutato? Chi ha concesso l'eccezione? Che cosa è stato negato?

Il report finale può essere utile per gli umani. Ma non può essere la fonte primaria della verità. La fonte primaria deve essere la traiettoria verificabile delle azioni.

Altrimenti il fallimento non puzza più di fallimento. Profuma di procedura.

La policy come comportamento atteso

La frase da portare avanti è semplice: la policy non è contesto. È comportamento atteso.

Questo non significa che ogni policy debba diventare codice rigido. Sarebbe ingenuo. Molte regole aziendali sono interpretative, incomplete, dipendenti dal caso concreto. Esistono zone in cui serve giudizio, e proprio per questo gli agenti possono essere utili.

Ma le policy critiche devono essere classificate. Non tutte hanno lo stesso peso operativo.

Tipo di policy	Scopo	Dove risiede	Effetto sull'azione
Orientamento	Aiuta l'agente a scegliere formato, tono, priorità o percorso consigliato.	Prompt o contesto, con valore consultivo.	Suggerisce, ma non deve bloccare da sola.
Soglia / vincolo	Blocca importi, contenuti, combinazioni di dati o operazioni sopra un limite definito.	Policy engine esterno o workflow deterministico.	Consente, nega o richiede prova prima del tool.
Mandato	Verifica se l'attore ha diritto ad agire per quello scopo e con quella durata.	Sistema di autorizzazione, legato a identità, task e scadenza.	Restringe strumenti, dati e deleghe disponibili.
Escalation	Impone arresto, prova aggiuntiva o intervento umano prima dell'esecuzione.	Control plane e sistema HITL tracciato.	Sospende l'azione finché la prova non esiste.
Divieto	Impedisce l'azione anche quando sarebbe tecnicamente possibile e apparentemente utile.	Regola statica non negoziabile.	Blocca sempre, anche se il modello insiste.

Trattarle tutte come testo recuperabile significa perdere la differenza più importante: non tutte le informazioni hanno lo stesso rango operativo.

Una nota di procedura può aiutare l'agente a scegliere il formato corretto. Una soglia di spesa può bloccare l'azione. Una lista di ruoli autorizzati può determinare se il mandato esiste. Un vincolo normativo può vietare la combinazione di dati. Una clausola contrattuale può richiedere revisione umana prima dell'output.

L'architettura deve rappresentare queste differenze. Non basta che il modello le legga. Deve esserci un modo per trasformarle in decisioni applicabili: procedi, fermati, chiedi prova, limita il campo, riduci i permessi, registra eccezione, scala a un umano.

Qui il capitolo apre il resto dell'ebook.

Se la policy deve diventare comportamento, dobbiamo sapere dove si applica. Non può essere soltanto al login, perché l'agente non compie una sola azione. Non può essere soltanto all'accesso dell'applicazione, perché l'applicazione è un contenitore troppo largo. Non può essere soltanto alla fine, perché alla fine lo stato potrebbe essere già cambiato.

La decisione deve scendere più in basso.

Non solo: questo utente può entrare?

Ma: questa azione può essere eseguita, su questo dato, per questo mandato, in questo momento, con questa prova?

E questa domanda ci porta al secondo problema: il perimetro. Zero Trust ha insegnato alle aziende a non fidarsi della rete. Era necessario. Ma con gli agenti non basta più decidere chi entra. Bisogna decidere, azione per azione, che cosa può davvero accadere dopo l'ingresso.

Zero Trust non basta agli agenti

Il capitolo precedente lasciava una domanda in sospeso: se la policy deve comandare l'azione, dove si prende la decisione?

La risposta tradizionale dell'enterprise è stata: al confine.

Per molti anni quel confine ha avuto una forma abbastanza riconoscibile. Prima era la rete: dentro l'azienda eri in una zona relativamente fidata, fuori eri in territorio ostile. Poi quel modello ha iniziato a crollare sotto il peso di portatili, cloud, VPN, SaaS, dispositivi personali, fornitori, consulenti, fusioni, filiali, lavoro remoto e tutto il resto della normale poesia infrastrutturale aziendale.

Zero Trust nasce come correzione necessaria a quella nostalgia. La sicurezza aziendale ha passato più di dieci anni a ripetere una frase sensata: non fidarti della rete. Non chiedere soltanto se qualcuno è "dentro". Chiedi chi è, da quale dispositivo arriva, in quale contesto si trova, quale applicazione vuole usare, con quale livello di rischio.

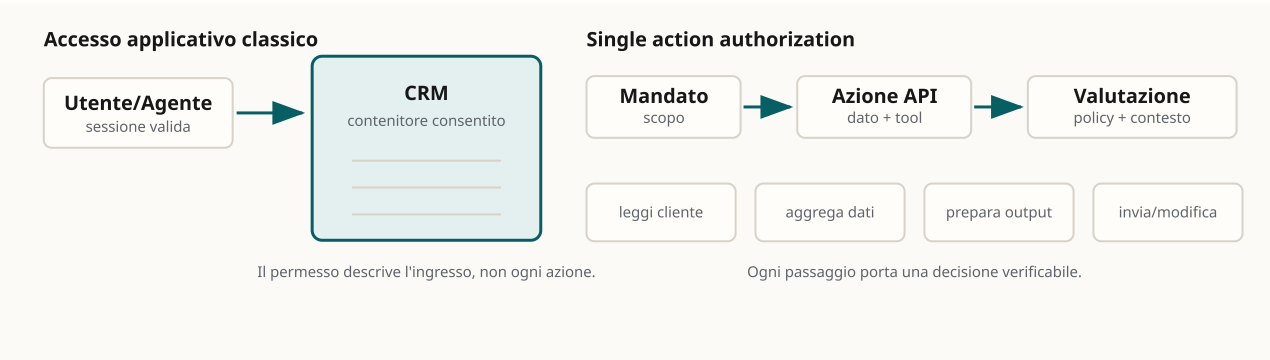
Era un progresso enorme. Il vecchio perimetro, quello con l'esterno cattivo e l'interno buono, era già una favola raccontata da firewall con molta autostima.

Ma gli agenti AI spostano di nuovo la domanda.

Non basta più chiedere: puoi usare questa applicazione?

Bisogna chiedere: puoi eseguire questa azione, su questo dato, per questo mandato, in questo momento?

Figura 2: dal perimetro applicativo all'azione



La domanda è diventata troppo larga

BeyondCorp, il modello raccontato da Google nel 2014, ha reso popolare una formulazione moderna di Zero Trust: niente intranet privilegiata, niente castello con fossato, accesso deciso in base a identità, dispositivo, contesto e policy, anche quando l'applicazione sta su Internet.

Nel mondo per cui quel modello è nato, l'attore principale era ancora una persona. Una persona apre il portatile, entra in un'applicazione, legge una pagina, scarica un documento, aggiorna una scheda cliente, magari fa una richiesta privilegiata. Anche quando i sistemi sono complessi, la velocità dell'azione resta grossomodo umana.

Un agente cambia il ritmo e la forma del problema.

Non perché sia magico. Perché non si stanca, non si distrae nel modo in cui si distrae una persona, non ha il pudore operativo di chi si accorge di fare qualcosa di strano. Se gli dai accesso a posta, documenti, CRM, richieste interne, repository e strumenti amministrativi, può attraversarli in sequenza a una velocità che rende ridicolo il controllo a posteriori.

Quando il log arriva all'analista, la parte interessante della storia potrebbe essere già successa.

Questo è il primo punto in cui molte implementazioni di Zero Trust mostrano il limite pratico. Decidono ancora al livello dell'applicazione o del servizio: puoi entrare in Drive, puoi usare il CRM, puoi interrogare quel sistema di supporto, puoi aprire quel repository. Dentro quel confine poi esistono ruoli, gruppi, permessi e controlli. Ma l'unità mentale rimane spesso l'applicazione.

Con un agente, l'applicazione è un confine troppo grande.

È come autorizzare qualcuno a entrare in un magazzino per prendere una vite, e poi stupirsi se attraversa tutte le corsie fotografando gli scaffali. Il problema non è che sia entrato. Il problema è che il permesso non descriveva l'azione.

Il buco si chiama autorità ambientale

Nel paper `Beyond Zero: Enterprise Security for the AI Era`, Joseph Valente e Michal Zalewski usano un concetto utile: ambient authority, autorità ambientale.

È il caso in cui un agente eredita, di fatto, i permessi troppo larghi dell'utente per conto del quale opera.

La trappola è elegante perché sembra ragionevole. Se Marco può leggere certi dati, perché l'assistente di Marco non dovrebbe leggerli per aiutarlo? Se Marco può usare un sistema interno, perché l'agente che Marco ha attivato non dovrebbe usarlo? Se Marco è autenticato, il problema sembra chiuso.

Non è chiuso. È appena iniziato.

Un essere umano porta con sé freni che non sono scritti in nessuna policy. Sa che quel documento non c'entra col progetto. Si accorge che sta aprendo la cartella sbagliata. Ricorda che quel cliente appartiene a un altro gruppo. Si ferma perché una richiesta suona socialmente strana prima ancora che tecnicamente vietata.

Non sempre, certo. Gli incidenti interni non li abbiamo inventati con l'AI. Però una parte della sicurezza aziendale vive da sempre dentro questa zona grigia: contesto implicito, abitudine, responsabilità, timore di essere scoperti, buon senso, vergogna professionale. Tutta roba difficilissima da mettere in YAML, e infatti il settore ci prova lo stesso con risultati spesso comici.

Un agente non possiede quella geografia implicita. Può simularla, se il sistema è progettato bene. Può ricevere istruzioni, leggere un piano, avere una propria identità. Ma se il controllo reale resta "agisce come Marco", l'organizzazione ha appena trasformato un permesso umano in una superficie automatizzabile.

Questa è la parte che rende gli agenti un problema enterprise vero, non un dettaglio di prodotto.

Non stiamo parlando della dimostrazione in cui un assistente prenota un viaggio o riassume una riunione. Parliamo di agenti che leggono dati commerciali, preparano offerte, interrogano sistemi di richiesta, aprono richieste, chiamano API, generano report, incrociano informazioni su clienti e fornitori.

Ogni azione, presa da sola, può sembrare legittima.

La sequenza può non esserlo.

> **Esempio operativo - aggregazione dati**
>
> Un agente commerciale prepara una proposta. Legge il listino autorizzato, recupera note tecniche accessibili, confronta offerte precedenti e aggiunge una sintesi di ticket aperti dal cliente. Ogni lettura è consentita. Il report finale, però, combina prezzi riservati, criticità operative e segnali di churn in un documento esportabile verso un partner esterno. Nessuna singola porta era proibita. È la combinazione a violare il perimetro.

Il rischio non è solo l'agente cattivo. È l'agente troppo obbediente dentro un perimetro troppo largo.

L'azione come nuova unità minima

La proposta più utile di Beyond Zero è spostare il confine di fiducia dall'applicazione alla singola azione su una singola risorsa.

Non "puoi usare questo strumento".

Ma: puoi eseguire questa operazione su questo dato, con questa intenzione, in questo contesto?

Vale che l'accesso arrivi da interfaccia web, API, Model Context Protocol (MCP) o altri canali. Il canale non deve essere il punto centrale. Il punto centrale è l'azione che produce stato.

Questa è la continuità naturale con il capitolo precedente. Se la policy non è contesto ma comportamento atteso, allora il controllo deve applicarsi nel momento in cui il comportamento sta per accadere. Non solo all'inizio della sessione. Non solo al login. Non solo quando l'utente apre l'applicazione.

Al momento dell'azione.

Autorizzazione non significa più soltanto: questo utente può entrare?

Significa: questa azione è coerente con il mandato? Il dato appartiene al perimetro corretto? Il ruolo dell'utente basta davvero? L'agente sta agendo dentro il compito assegnato o ha allargato il giro? La combinazione di dati cambia il rischio? Serve una prova? Serve escalation? Lo strumento deve poter essere chiamato o deve restare chiuso?

Il controllo non può più essere pensato come una barriera che ogni tanto qualcuno attraversa. Diventa un circuito ad alta frequenza.

Questa frase è importante, ma non va trasformata in feticcio tecnologico. Non significa che ogni decisione debba essere affidata a un modello opaco che ragiona in tempo reale su tutto. Sarebbe solo un modo più moderno per costruire un oracolo non verificabile.

La distinzione sana è tra pavimento e soffitto.

Il pavimento è fatto di regole statiche, verificabili, noiose e necessarie: divieti, soglie, ruoli, classificazione dei dati, limiti di strumento, separazioni di funzione, azioni sempre proibite. Il soffitto è dinamico: contesto del lavoro, comportamento recente, coerenza con il compito, rischio, anomalie, richiesta di prova aggiuntiva.

Un modello solo statico non vede abbastanza. Un modello tutto dinamico vede troppo e spiega troppo poco.

Il punto è usare regole statiche come base e ragionamento dinamico come controllo contestuale, dentro una macchina di autorizzazione che resti verificabile.

Non AI al posto della sicurezza.

AI dentro un sistema di sicurezza che può ancora essere interrogato, contestato e corretto.

Il perimetro non scompare, si frantuma

Dire che l'applicazione è un confine troppo grande non significa che il perimetro sparisca. Questa è una delle semplificazioni peggiori del discorso su Zero Trust: siccome il vecchio perimetro non basta, allora il perimetro non esiste più.

No. Il perimetro non scompare: si frantuma.

- perimetro dell'identità: chi dichiara di essere;
- perimetro del dispositivo e della sessione: da dove opera;
- perimetro del dato: quale informazione sfiora, legge o combina;
- perimetro dello strumento: quale API, MCP o altro canale chiama;
- perimetro del mandato: qual è lo scopo esplicito e delimitato;
- perimetro della singola azione: cosa modifica effettivamente;
- perimetro dell'output: cosa sintetizza, invia o esporta.

Gli agenti attraversano questi perimetri con naturalezza. È la loro utilità. Prendono uno scopo e lo traducono in passaggi. Solo che ogni passaggio può spostare informazioni, cambiare priorità, creare artefatti, attivare altri sistemi, coinvolgere altre persone.

Il vecchio controllo applicativo vede spesso una storia grossolana: utente autenticato, applicazione consentita, operazione riuscita.

L'architettura agentica deve vedere una storia più fine: utente, agente, mandato, strumento, dato, azione, prova, risultato.

Questo non è dettaglio burocratico. È ciò che permette di distinguere un agente utile da una scorciatoia con accessi ereditati.

Un agente che legge un documento per riassumerlo sta facendo una cosa. Un agente che legge cinque documenti, combina dati commerciali e produce una proposta per un cliente fuori perimetro ne sta facendo un'altra. Un agente che prepara una bozza e la lascia in revisione è diverso da uno che invia direttamente il messaggio. Un agente che apre una richiesta interna è diverso da uno che cambia la priorità di una richiesta esistente.

Se il sistema vede tutte queste cose come "Marco ha usato l'applicazione", non ha abbastanza granularità per governare agenti.

Ha ancora Zero Trust, ma non abbastanza Zero.

Il prezzo del controllo dinamico

Beyond Zero introduce anche l'idea di un Enterprise Security World Model: una mappa viva dell'organizzazione usata per decidere chi può fare cosa. Ruoli, dati, incarichi, relazioni, strumenti, agenti, comportamenti normali, anomalie, rischi.

La direzione è comprensibile. Se vuoi autorizzare azioni a livello fine, ti serve contesto. Devi sapere non solo chi chiama, ma perché chiama, su cosa sta lavorando, quali dati sta toccando, quale agente opera per conto di chi, quale sequenza è plausibile e quale no.

Ma questa mappa è potentissima. E proprio per questo è pericolosa.

Se è sbagliata, blocca lavoro legittimo. Se è incompleta, lascia passare azioni rischiose. Se è troppo aggressiva, trasforma l'azienda in un aeroporto permanente. Se è troppo permissiva, diventa un altro cruscotto da mostrare con orgoglio finché non serve davvero. Se è opaca, il team di sicurezza non riesce a spiegare perché una decisione sia stata presa. Se è manipolabile, l'attaccante non deve più forzare ogni applicazione: deve influenzare il modello che decide l'accesso.

La sicurezza a livello di azione non è gratis.

Richiede classificazione dei dati, identità degli agenti, attribuzione delle azioni, telemetria affidabile, integrazione con sistemi HR e sistemi di progetto, bassa latenza, policy scritte bene, flussi di eccezione, controllo dei falsi positivi, log leggibili e responsabilità chiare.

Se manca metà di questa roba, "autorizzazione dinamica" rischia di diventare un nome elegante per "abbiamo aggiunto un altro sistema che nessuno capisce".

Questo è il punto da tenere fermo. Non basta dire che Zero Trust va superato. Bisogna evitare che il superamento diventi una scatola nera più potente del problema che voleva risolvere.

Cosa cambia per un'architettura enterprise

Il valore di questo capitolo non è proporre a ogni azienda di implementare domani mattina una versione completa di Beyond Zero. Sarebbe ingenuo, costoso e probabilmente anche un ottimo modo per far odiare la sicurezza a chi già la tollera con fatica.

Il valore è costringere l'architettura a fare domande più precise.

Quali azioni dei nostri agenti sono attribuibili a un utente, a un agente e a un compito?

Quali strumenti interni ragionano ancora solo per accesso applicativo?

Quali dati sono classificati abbastanza bene da sostenere policy reali?

Quando un agente esporta, riassume o incrocia dati, il sistema vede tre azioni distinte o solo un utente autenticato che ha fatto qualcosa?

Chi approva l'eccezione?

Chi risponde se il ragionamento dinamico sbaglia?

Queste domande non richiedono per forza un prodotto nuovo. Richiedono prima di tutto di smettere di fingere che il login sia il posto in cui si risolve il problema.

Zero Trust non è morto. È diventato insufficiente come unità di misura.

Ha tolto fiducia alla rete, ed era necessario. Ora bisogna togliere fiducia anche all'applicazione come confine naturale, all'identità umana come contenitore implicito di autorità e al controllo a posteriori come garanzia.

Gli agenti non chiedono solo più permessi. Chiedono una sicurezza capace di seguire l'azione mentre accade.

Il problema non è più sapere se qualcuno è autorizzato a entrare. È sapere che cosa sta facendo, per conto di chi, su quale dato, con quale scopo, e chi paga quando rompe.

Ed è qui che il capitolo successivo diventa inevitabile. Se l'azione diventa l'unità minima del controllo, resta da capire quale autorità accompagna quell'azione. Dell'utente? Dell'agente? Del processo? Del modello? Del sub-agente che ha ricevuto un pezzo del lavoro?

Senza questa distinzione, l'identità dell'agente serve a poco. Un badge dice chi sta chiamando. Non dice ancora chi gli ha dato il potere di farlo, per quale compito, con quali limiti e fino a quando.

L'autorità degli agenti non è identità

Il capitolo precedente chiudeva con una domanda semplice solo in apparenza: se l'azione diventa l'unità minima del controllo, quale autorità accompagna quell'azione?

È la domanda che molte architetture agentiche cercano di evitare con una scorciatoia familiare: diamo un'identità all'agente.

Il modo più veloce per sbagliare la sicurezza degli agenti AI è dare loro un badge e pensare di aver risolto il problema.

È una tentazione comprensibile. Per trent'anni l'informatica aziendale ha trasformato quasi ogni domanda di sicurezza in una domanda di identità: chi sei, a quale gruppo appartieni, quale ruolo hai, quale token presenti, quale dispositivo stai usando. Quando arriva un nuovo attore nel sistema, la reazione naturale è iscriverlo nello stesso registro. Anche l'agente deve avere un'identità. Fine della riunione, tutti a casa, possibilmente prima che qualcuno nomini OAuth.

Peccato che l'agente non sia una persona con continuità biologica, nè una macchina con comportamento deterministico, nè un servizio statico con uno scopo stabile. È un'entità operativa che può nascere per un compito, usare strumenti, leggere dati, delegare pezzi di lavoro, sintetizzare risultati e sparire.

Il punto non è soltanto riconoscerla.

Il punto è seguire l'autorità che porta con sé mentre si muove.

Il badge non basta

L'identità risponde a una domanda necessaria: chi sta chiamando?

Ma gli agenti costringono l'architettura a fare una domanda più scomoda: chi ha dato a questa cosa il potere di chiamare, per quale compito, con quali limiti e fino a quando?

Senza questa distinzione, il badge diventa teatro. Un agente può essere perfettamente autenticato e comunque agire fuori mandato. Può presentare un token valido e usare un contesto manipolato. Può chiamare uno strumento consentito con argomenti non consentiti. Può delegare un sotto-compito apparentemente innocuo che ricompone a valle ciò che nessuna singola chiamata avrebbe dovuto esporre.

Immagina un agente commerciale incaricato di preparare una proposta per un cliente. Legge il CRM, apre vecchie offerte, consulta note tecniche, chiede a un altro agente di stimare i costi, riassume contratti simili, produce una bozza. Quale identità stai controllando? L'utente che ha chiesto il lavoro? L'agente principale? Il sub-agente che ha interrogato i documenti legali? Il processo che ha chiamato l'API? Il modello che ha generato la sintesi?

La risposta "sì" a tutte queste domande non è una policy: è un incidente in forma di architettura.

Qui serve una distinzione minima, senza trasformare il capitolo in glossario:

Entità / livello	Domanda a cui risponde	Ciclo di vita	Rischio se errato
Identità umana	Chi ha avviato o richiesto il lavoro?	Lunga, legata alla persona e ai ruoli aziendali.	L'agente eredita privilegi umani troppo larghi.
Identità agente	Quale soggetto operativo sta eseguendo il compito?	Temporanea o persistente, ma distinta dall'utente.	Le azioni diventano indistinguibili da quelle dell'utente.
Workload	Quale processo tecnico chiama API e strumenti?	Runtime, container, job o servizio.	L'account di servizio viene scambiato per mandato.
Mandato	Perché questa azione è legittima ora?	Breve, vincolato a scopo, dato e durata.	Un compito limitato diventa accesso generale.
Delega	Quale quota di autorità passa a valle?	Per sotto-compito, attenuata e tracciata.	Il sub-agente riceve più potere del necessario.
Prova	Come ricostruiamo la decisione?	Permanente quanto serve ad audit e contestazione.	Resta solo il racconto dell'agente, non evidenza.

Se questi livelli finiscono nello stesso account di servizio, l'azienda non ha semplificato l'architettura. Ha nascosto il problema sotto un nome solo.

Anche lasciare le chiavi sotto lo zerbino è comodo.

Identità e autorità non coincidono

Un'identità dice che un attore è riconoscibile.

L'autorità dice che cosa può fare, per conto di chi, dentro quale confine.

La differenza sembra sottile finché l'agente non inizia a lavorare. Poi diventa tutto.

Un utente può avere diritto a leggere un documento, ma non a farlo leggere a un agente generico. Un agente può avere diritto a sintetizzare un contratto, ma non a incrociarlo con dati commerciali di un altro cliente. Un sub-agente può avere diritto a stimare un costo, ma non a vedere l'intera trattativa. Un workload può avere accesso tecnico a un'API, ma non autorità organizzativa per usarla in quel contesto.

L'identità è un attributo del chiamante.

L'autorità è una proprietà della relazione tra chiamante, mandato, dato, strumento e azione.

Questo è il punto in cui l'architettura deve smettere di ragionare solo per ruoli statici. Non perché i ruoli siano inutili. Sono necessari. Ma non bastano a descrivere una catena agentica.

Se Marco avvia un agente per preparare una bozza, l'agente non deve ereditare automaticamente tutto ciò che Marco può fare nei sistemi aziendali. Deve ricevere il minimo necessario per quel compito, per quel periodo, con quei dati e con quegli strumenti.

Ogni passaggio deve restringere il potere, non allargarlo.

Questa è la regola che molte implementazioni violano appena incontrano la produzione. L'agente nasce limitato, poi serve un documento in più. Poi serve un sistema in più. Poi serve un'integrazione provvisoria. Poi il provvisorio resta. Dopo tre sprint, il sistema ha inventato un permesso implicito, solo con una dashboard più moderna.

Non è progresso. È debito di autorizzazione con interfaccia conversazionale.

La delega deve restringersi

La delega è inevitabile. Un agente utile non esegue sempre tutto da solo. Chiama strumenti, invoca servizi, chiede a un altro agente di risolvere un segmento, recupera dati, sintetizza, passa risultati ad altri sistemi.

Il problema non è la delega.

Il problema è la delega che allarga.

In un'architettura sana, ogni delega dovrebbe portare con sé meno potere della richiesta originale. Se il compito è "prepara una bozza di proposta per il cliente X", il sub-agente che stima i costi non dovrebbe poter leggere tutto il CRM. Il componente che sintetizza clausole legali non dovrebbe poter inviare email. Lo strumento che recupera documenti non dovrebbe poter modificare priorità di una richiesta interna.

Sembra ovvio. Infatti è esattamente il tipo di ovvietà che i sistemi reali tradiscono quando conviene usare un token già disponibile.

L'autorizzazione non è più una decisione puntuale. È una proprietà del flusso di lavoro.

Se un agente legge un documento A e un documento B, entrambi accessibili singolarmente, può produrre una sintesi che viola una policy di aggregazione? Se delega un sotto-compito, il sub-agente riceve solo la quota di autorità necessaria o eredita un permesso più largo? Se il lavoro dura trenta minuti, l'autorizzazione valida all'inizio vale ancora alla fine? Se un accesso viene scoperto come non valido, quali risultati a valle sono contaminati?

Queste non sono finezze accademiche. Sono il cuore del problema enterprise.

Molte violazioni non nascono da una singola porta aperta, ma dalla combinazione legittima di porte che nessuno aveva pensato insieme. L'agente è bravo proprio in questo: combinare, attraversare, sintetizzare. Se il sistema di autorizzazione vede soltanto le singole aperture, perde la forma dell'azione.

Propagazione, aggregazione, tempo

Il nodo vero si può descrivere con tre parole: propagazione, aggregazione, tempo.

Il triplo nodo

Propagazione	Aggregazione	Tempo
chi passa il permesso a chi?	cosa succede quando dati innocui, messi insieme, diventano sensibili?	per quanto resta valida una decisione?

Tempo: per quanto resta valida una decisione?

Un agente che lavora su più passaggi non produce solo chiamate isolate. Produce una catena. Recupera un dato, lo trasforma, lo consegna a un altro componente, lo incorpora in un output, lo usa come base per una decisione successiva. A ogni passaggio, l'autorità dovrebbe restare leggibile.

Se un dato non può uscire da un certo contesto, quella restrizione deve seguire anche la sintesi. Non può evaporare perché il testo è stato riassunto. Se un agente lavora per un utente, il mandato deve essere visibile lungo tutta la catena. Se un sub-agente sbaglia, deve essere possibile capire quale decisione lo ha abilitato. Se una combinazione di dati viola una policy, il sistema deve vederla prima che diventi un report elegante in una cartella condivisa.

La sicurezza tradizionale tende a pensare in termini di muro e porta.

La sicurezza degli agenti deve pensare in termini di catena di custodia.

Non basta sapere chi ha aperto la porta. Devi sapere che cosa ha preso, a chi lo ha passato, come è stato trasformato e se il risultato finale porta ancora le stesse restrizioni.

Questo cambia anche il logging. Un log che dice "agente X ha chiamato strumento Y" è meglio di niente, ma non basta. Serve sapere quale mandato giustificava la chiamata, quale dato è stato toccato, quale policy è stata valutata, quale prova è stata prodotta, quale output ha ereditato quei vincoli.

Altrimenti hai una cronologia tecnica. Non una prova di governance.

Hai una fede con logging.

La decisione deve stare fuori dall'agente

A questo punto torna un principio già emerso nel capitolo sulla policy: la parte normativa non può vivere solo dentro il modello.

Non chiedere all'agente di decidere da solo se una chiamata è sicura. L'agente è proprio la parte che può essere confusa dal contesto. Un documento, una pagina, un messaggio, un risultato intermedio possono falsificare l'apparenza dell'autorità. Possono convincere l'agente che una certa azione sia richiesta, legittima, urgente.

Se il controllo resta dentro l'agente, il sistema chiede alla parte più manipolabile di proteggere il confine più delicato.

È un'idea pessima con un buon pitch.

La direzione sana è spostare la decisione in un punto esterno: gateway, motore di autorizzazione, broker di strumenti, livello di policy, chiamalo come vuoi. Il nome conta meno della proprietà architetturale: l'agente non deve poter leggere, riscrivere o negoziare da solo le regole che lo limitano.

Quando l'agente chiede di usare uno strumento, il sistema esterno deve poter valutare identità, mandato, argomenti, dato, contesto e rischio. Deve poter dire sì. Deve poter dire no. Deve poter chiedere prova. Deve poter imporre escalation. Deve poter lasciare una traccia verificabile.

L'impresa non ha bisogno solo di permessi.

Ha bisogno di prove.

Questa è la differenza tra un'architettura che spera e un'architettura che può essere interrogata.

Il controllo che sopravvive

La domanda seria, quindi, non è "come diamo identità agli agenti?".

Quella è la parte facile, o almeno la parte vendibile.

La domanda seria è: quale controllo sopravvive quando l'agente delega, aggrega, sintetizza e produce risultati che altri useranno come input?

Un controllo sopravvive se resta attaccato all'azione anche dopo la prima chiamata. Sopravvive se il mandato segue il flusso. Sopravvive se la delega restringe. Sopravvive se le sintesi ereditano vincoli. Sopravvive se la revoca ha conseguenze sui risultati a valle. Sopravvive se l'audit non guarda solo il racconto finale dell'agente, ma la catena delle decisioni che ha permesso quel racconto.

Questo è scomodo perché obbliga le aziende a guardare ciò che spesso fingono di avere già: classificazione dati decente, responsabilità reali, policy granulari, registri utilizzabili, separazione tra identità umana e autorità delegata, revoca leggibile, eccezioni governate.

Gli agenti non inventano questo debito.

Lo rendono eseguibile.

Il punto forte, quindi, è questo: l'identità è il nome sull'etichetta. L'autorità è la sostanza pericolosa dentro il contenitore.

Se l'agente ha un nome ma porta in giro permessi larghi, deleghe implicite e sintesi senza provenienza, non abbiamo migliorato la sicurezza. Abbiamo solo dato un badge a qualcosa che corre più veloce di noi.

Ed è qui che il capitolo successivo diventa necessario. Se l'autorità deve seguire il flusso, il controllo deve muoversi alla stessa velocità dell'agente. Ma un controllo che decide troppo in fretta, su troppo contesto e con troppa opacità, può diventare un nuovo potere aziendale non governato.

Il problema non è soltanto costruire un sistema che controlla gli agenti.

È controllare il sistema che controlla.

La sicurezza a velocità macchina ha un prezzo

La promessa più seducente della sicurezza moderna è anche la più pericolosa: decidere prima che l'umano capisca cosa sta succedendo.

Con gli agenti AI questa promessa smette di essere una fantasia da fornitore. Diventa quasi un requisito operativo.

Se un sistema automatico può leggere migliaia di documenti, chiamare strumenti, aprire richieste, interrogare repository, confrontare contratti, produrre sintesi e attivare altri sistemi a velocità macchina, non puoi difenderti con un controllo pensato per la velocità umana. L'analista che guarda il log dopo pranzo arriva tardi. Il comitato che approva l'eccezione giovedì prossimo arriva ridicolo. Il modello in cui prima lasciamo fare, poi controlliamo funziona solo finché il danno corre più lentamente dell'organigramma.

I capitoli precedenti hanno spostato il problema sempre più vicino all'azione. La policy non può restare contesto. Il perimetro applicativo è troppo grande. L'identità non basta, perché l'autorità deve seguire mandato, delega, prova e trasformazioni a valle.

Adesso arriva la conseguenza scomoda: se l'agente lavora veloce, anche il controllo deve lavorare veloce.

Fin qui, tutto giusto.

Il problema è il prezzo.

Per difendersi a velocità macchina, l'azienda deve costruire un nuovo piano di controllo della sicurezza. Un sistema che osserva, interpreta, classifica, interrompe, contiene, chiede prove e produce decisioni. Chiamarlo "motore dinamico di autorizzazione" suona meno inquietante. Ma la sostanza non cambia: stiamo creando un nuovo control plane.

E ogni control plane prima o poi diventa politico, anche quando è scritto in Go e ha una dashboard bellissima.

Il controllore non è neutro

Beyond Zero, la proposta di Alphabet Security per estendere Zero Trust all'era degli agenti, prende atto di questo cambio di ritmo. Non basta autorizzare l'accesso a un'applicazione. Bisogna decidere su azioni, risorse, contesto, rischio, comportamento, intenzione e mitigazioni disponibili. E bisogna farlo abbastanza in fretta da non trasformare la sicurezza in un passaggio burocratico che gli agenti aggirano per semplice pressione operativa.

La parte più interessante non è lo slogan. È l'architettura implicita.

Da una parte ci sono policy statiche: regole dure, verificabili, comprensibili, adatte a dire che alcune cose non si fanno. Dall'altra c'è un motore dinamico: un sistema che osserva il contesto e decide se consentire, bloccare, chiedere prova, ridurre il permesso, imporre attrito o scalare a una persona.

La distinzione è sensata. Le policy statiche sono stabili, ma spesso cieche. Il ragionamento dinamico vede di più, ma rischia di diventare opaco. La direzione sana non è scegliere una delle due parti. È tenerle in tensione: pavimento normativo sotto, valutazione contestuale sopra.

Il guaio comincia quando il secondo livello viene trattato come se fosse neutro.

Non lo è.

Un sistema che decide in tempo reale se un agente può leggere un dato, sintetizzarlo, inviarlo, combinarlo o usarlo come base per un'altra azione non è un filtro tecnico. Sta distribuendo potere operativo dentro l'organizzazione. Stabilisce che cosa conta come rischio, che cosa conta come comportamento normale, quali anomalie meritano attrito, quali eccezioni sono tollerabili, quali attori devono fermarsi.

Questa non è una piccola estensione di Zero Trust.

È il sistema nervoso dell'azienda.

Il world model sporco decide male

Per funzionare, un controllo dinamico deve avere una rappresentazione computabile dell'organizzazione. Chi lavora su cosa. Chi è responsabile di quale cliente. Quali agenti operano per conto di quali persone. Quali dati sono sensibili. Quali relazioni contano. Quali strumenti sono coerenti con un mandato. Quali azioni sono normali in quel contesto.

Sulla carta è elegante.

Nella realtà aziendale, questa mappa spesso nasce sopra dati amministrativi sporchi: ruoli non aggiornati, responsabilità fittizie, gruppi ereditati da migrazioni antiche, progetti chiusi ma ancora attivi nei sistemi, cartelle condivise che nessuno osa toccare perché "sono sempre state così", eccezioni temporanee che hanno ottenuto la cittadinanza permanente.

Se costruisci un world model sopra questa palude, non ottieni una mappa.

Ottieni una palude navigabile via API.

Questo è il punto che molti discorsi sugli agenti saltano. Il contesto non è neutro. Quando lo usi per decidere un accesso, il contesto diventa comportamento autorizzato o negato.

Se il sistema crede che un utente lavori ancora su un progetto chiuso, autorizza troppo. Se non vede una collaborazione reale, blocca lavoro legittimo. Se il dato è classificato male, prende decisioni raffinate su una bugia. Se il mandato è scritto in modo vago, concede all'agente spazio interpretativo proprio dove serviva vincolo.

Un controllo dinamico è potente solo quanto è affidabile il mondo che gli hai dato da interpretare.

Questo rende noiosa ma centrale una parte del lavoro che di solito non finisce nelle demo: classificazione dati, responsabilità reali, pulizia dei gruppi, ciclo di vita dei progetti, revoca degli accessi, scadenza dei mandati, registri coerenti, eccezioni governate.

La sicurezza agentica non elimina l'igiene enterprise.

La rende obbligatoria.

Supervisori che giudicano agenti

C'è poi un secondo livello: usare sistemi AI per sorvegliare altri sistemi AI.

L'idea è comprensibile. Se gli agenti generano azioni a velocità macchina, un controllo interamente umano non scala. Per azioni a basso rischio puoi analizzare dopo, correggere traiettorie, migliorare regole. Per azioni ad alto rischio devi prevenire prima che il danno avvenga. Serve copertura, capacità di intercettazione e tempo di risposta compatibile con il rischio.

In altre parole: non basta dire "monitoriamo".

Devi sapere quanto monitori, quanto intercetti, quanto rapidamente reagisci e che cosa succede quando il supervisore sbaglia.

Qui il rovescio è evidente. Se metti un modello a giudicare un altro modello, hai creato una gerarchia di opacità. Può essere necessaria. Può essere l'unico modo realistico per scalare. Ma non è gratis.

Devi verificare il supervisore. Devi misurare i suoi errori. Devi decidere quando può bloccare da solo e quando deve chiamare una persona. Devi impedire che diventi un oracolo intoccabile perché "lo dice il sistema".

Chi ha lavorato in azienda conosce già la scena. Un sistema blocca qualcosa, nessuno sa esattamente perché, tutti aprono una richiesta, qualcuno cerca l'eccezione, il lavoro si ferma, e alla fine la regola viene allargata perché altrimenti il reparto urla.

Ora immagina la stessa scena a velocità macchina, con agenti che generano centinaia di azioni e un supervisore dinamico che applica attrito in tempo reale.

Se la governance non è chiara, non hai sicurezza.

Hai burocrazia automatizzata.

Il falso positivo è una scelta di governo

Nel lessico tecnico un falso positivo è un evento classificato come rischio quando non lo era.

Sembra un problema statistico.

In azienda è anche una decisione politica.

Se blocchi un agente che stava preparando una proposta importante, hai scelto sicurezza contro velocità commerciale. Se permetti un'esportazione sospetta per non disturbare il lavoro, hai scelto produttività contro contenimento. Se chiedi approvazione umana per ogni azione ambigua, hai

scelto controllo contro autonomia. Se automatizzi il blocco per le azioni ad alto rischio, hai scelto prevenzione contro contestabilità immediata.

Non esiste una soglia neutra.

Esistono soglie che qualcuno deve assumersi la responsabilità di impostare.

Gli agenti rendono questi compromessi meno episodici. Una persona può generare dieci decisioni dubbie in una giornata. Un agente può generarne migliaia in pochi minuti. Ogni soglia sbagliata si amplifica. Ogni dato organizzativo sporco diventa un input decisionale. Ogni eccezione concessa per comodità può diventare una corsia automatizzata.

Il punto non è rendere il controllo morbido per non disturbare.

Il punto è rendere il controllo governabile.

Una soglia deve avere un proprietario. Una regola deve avere una ragione. Un blocco deve essere contestabile. Un'eccezione deve scadere. Un falso positivo deve produrre apprendimento, non solo fastidio. Un falso negativo deve lasciare una lezione verificabile, non una riunione piena di aggettivi.

Questo è il prezzo adulto dell'automazione.

Chi controlla il controllore

La domanda seria, quindi, non è se serva un controllo dinamico sugli agenti. Serve.

La domanda seria è: chi controlla il sistema che controlla?

Un control plane di sicurezza per agenti non può essere una scatola nera opaca. Deve possedere almeno quattro proprietà fondamentali.

I quattro requisiti del control plane agentico

Spiegabile:	definizione: ricostruisce la catena causale. Implicazione: una contestazione deve vedere dato, policy, soglia e motivo del blocco.
Verificabile:	definizione: regge una revisione tecnica, organizzativa e normativa. Implicazione: il controllore non può chiedere fiducia speciale perché è automatico.
Correggibile:	definizione: distingue errore di policy, dato sporco, modello sbagliato, mandato impreciso o comportamento anomalo. Implicazione: il falso positivo produce apprendimento, non solo eccezioni permanenti.
Limitato:	definizione: resta vincolato da regole statiche non negoziabili. Implicazione: il ragionamento dinamico non può riscrivere da solo i principi di sicurezza.

La parola "spiegabile" qui non significa chiedere al sistema un riassunto rassicurante. Significa poter ricostruire la catena: quale dato organizzativo ha pesato, quale policy è stata applicata, quale rischio è stato rilevato, quale alternativa era disponibile, quale soglia ha fatto scattare una verifica o un contenimento.

La parola "correggibile" significa che quando il sistema sbaglia non basta cliccare "consenti sempre". Serve capire se era sbagliata la policy, il dato, il modello del progetto, la classificazione della risorsa o il comportamento dell'agente. Altrimenti ogni falso positivo risolto male diventa un falso negativo futuro.

La parola "limitato" è forse la più importante. Un motore dinamico può aiutare a prendere decisioni fini. Non deve poter riscrivere da solo i principi di sicurezza. Il pavimento statico serve proprio a questo: alcune cose non si fanno, anche se il contesto sembra favorevole; alcune prove sono obbligatorie, anche se il modello è convinto; alcune azioni richiedono responsabilità umana, anche se rallentano.

> **Warning - Alert fatigue automatizzata**
>
> Un supervisore troppo sensibile non aumenta automaticamente la sicurezza. Se blocca o scala ogni ambiguità, produce approvazioni automatiche travestite da controllo umano. Se invece viene addolcito per ridurre il fastidio, rischia di trasformare i falsi positivi di oggi nei falsi negativi di domani. La soglia non è un dettaglio statistico: è una decisione di governo operativo.

Un controllore senza limiti non è governance.

È solo un'altra automazione che pretende fiducia.

Il prezzo giusto

La sicurezza a velocità macchina non è una moda. È la conseguenza naturale di agenti che operano a velocità macchina. Pensare di controllarli con processi umani tradizionali è rassicurante come mettere un vigile urbano davanti a un pacchetto UDP.

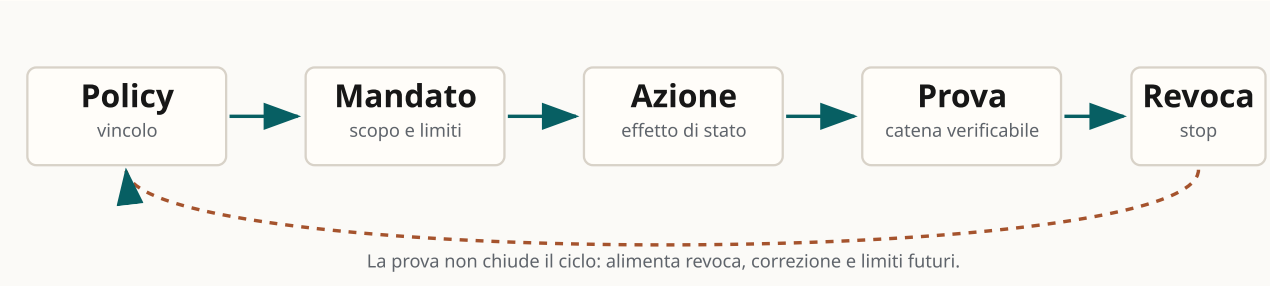
Ma il prezzo va detto bene.

Per avere autorizzazione dinamica servono dati organizzativi puliti, responsabilità reali, classificazione utile, telemetria affidabile, registri leggibili, soglie governate, ruoli chiari, procedure di contestazione e capacità di spegnere o contenere senza distruggere il lavoro legittimo.

Manca una di queste cose e il sistema si deforma.

Il rischio non è solo che l'agente sbagli. È che il sistema costruito per fermarlo diventi un altro centro di potere opaco, impossibile da discutere perché troppo complesso, troppo veloce, troppo automatico.

Diagramma principale: ciclo minimo di governance agentica



La frase "lo ha deciso la sicurezza" è già abbastanza fastidiosa quando dietro c'è una persona. Quando dietro c'è un motore dinamico alimentato da un world model aziendale, diventa una forma di governo.

Gli agenti obbligano la sicurezza enterprise a diventare più granulare, più contestuale e più veloce. Bene. Ma se il nuovo control plane non è verificabile, contestabile e governato, rischia di fare ciò che facevano già i vecchi sistemi opachi: proteggere l'organizzazione rendendola incapace di capire chi ha deciso cosa.

Il compito non è scegliere tra agenti liberi e sicurezza paralizzante.

Il compito è stabilire una soglia minima sotto la quale un agente non dovrebbe agire su processi critici.

Senza quella soglia, non stai costruendo architettura agentica enterprise.

Stai accelerando un atto di fede.

Sette principi minimi per agenti enterprise

Un agente AI in azienda non è pericoloso perché parla.

È pericoloso perché può agire.

Questa è la distinzione che attraversa tutto l'ebook. Finché l'AI resta interfaccia, il problema principale è la qualità della risposta. Quando diventa agente, il problema diventa la qualità dell'azione: quale sistema tocca, quale dato legge, quale decisione prepara, quale strumento invoca, quale output lascia dietro di sé.

Molte architetture enterprise stanno cercando di assorbire gli agenti con categorie vecchie: prompt, ruolo, token, applicazione, log. Servono ancora, ma non bastano. Il prompt orienta, ma non governa. Il ruolo riconosce, ma non delimita il mandato. Il token apre una porta, ma non spiega cosa l'agente sta facendo dentro la stanza. Il log racconta che qualcosa è successo, ma non sempre dimostra che doveva poter succedere.

Se c'è una frase da portare via, è questa: un agente enterprise non deve solo essere utile. Deve essere interrogabile.

Devi poter chiedere perché ha agito, per conto di chi, con quale autorità, su quale dato, usando quale strumento, secondo quale policy, con quale prova, e cosa succede se quella decisione viene revocata.

Senza queste risposte non hai architettura.

Hai automazione con una buona interfaccia.

1. La policy critica non vive nel prompt

Le istruzioni nel prompt possono orientare un agente. Possono dargli tono, priorità, preferenze, stile operativo. Possono ridurre errori banali. Possono anche aiutare il modello a comportarsi meglio nella maggior parte dei casi.

Ma una policy critica non può dipendere dalla buona memoria del modello.

Se una regola deve bloccare un'azione, chiedere una prova, imporre escalation, limitare un dato o vietare un output, quella regola deve vivere in un punto esterno, verificabile e applicabile. Il modello può conoscerla. Non deve possederla. Non deve poterla reinterpretare quando il contesto diventa rumoroso, urgente, manipolato o semplicemente troppo lungo.

Il criterio minimo è semplice: se la violazione di una policy produce danno reale, la policy non può essere soltanto testo nel contesto.

Deve diventare comportamento imposto dal sistema.

2. L'identità non basta: serve autorità esplicita

Dare un'identità a un agente è necessario. Non è sufficiente.

Un agente autenticato può comunque agire fuori mandato. Può usare un token valido per una richiesta sbagliata. Può ereditare troppo dall'utente che lo ha avviato. Può combinare accessi legittimi in un risultato illegittimo. Può delegare un pezzo di lavoro a un componente che non dovrebbe vedere tutto il contesto originale.

L'identità risponde alla domanda: chi sta chiamando?

L'autorità risponde a una domanda più difficile: perché questa chiamata è legittima, in questo momento, per questo compito, su questo dato?

Il criterio minimo: ogni azione significativa deve portare con sé identità, mandato, scopo, durata, limiti e responsabilità. Se questi elementi non sono distinguibili, l'azienda non sta autorizzando agenti. Sta dando badge a processi che non sa interrogare.

3. Ogni delega deve restringere il potere

Un agente utile delega. Chiama strumenti, invoca servizi, chiede aiuto ad altri agenti, spezza un compito in passaggi più piccoli.

La delega non è il problema.

Il problema è la delega che allarga.

Ogni passaggio dovrebbe portare meno potere della richiesta originale, non di più. Un sub-agente che stima costi non deve leggere tutto il CRM. Uno strumento che recupera documenti non deve poter modificare una pratica. Un componente che sintetizza clausole non deve poter inviare email. Un processo tecnico non deve trasformare accesso infrastrutturale in autorità organizzativa.

Il criterio minimo: se un agente delega, il sistema deve poter dimostrare che la delega era attenuata, limitata e tracciata.

Se la delega aumenta il potere disponibile, hai creato autorità ambientale.

E l'autorità ambientale prima o poi viene usata.

4. L'autorizzazione segue il flusso

Autorizzare l'inizio del lavoro non basta.

Un agente non produce una singola chiamata. Produce una catena: legge, trasforma, combina, delega, sintetizza, esporta, prepara decisioni, alimenta altri sistemi. Ogni passaggio può cambiare il rischio.

Un dato innocuo può diventare sensibile quando viene aggregato. Un'informazione che poteva restare dentro un contesto può violare una policy quando finisce in un output. Una sintesi può perdere la provenienza e conservare comunque il vincolo. Una revoca può arrivare dopo l'inizio del lavoro e contaminare risultati già prodotti.

Il criterio minimo: mandato, vincoli e restrizioni devono propagarsi lungo il flusso.

Non basta controllare la porta d'ingresso. Devi seguire quello che l'agente prende, trasforma e lascia dietro di sé.

5. Il non fare è un risultato valido

Un agente enterprise non deve essere premiato solo perché completa compiti.

Deve essere progettato anche per fermarsi.

Fermarsi non è un errore quando manca autorità. Chiedere una prova non è un fallimento quando il rischio è ambiguo. Imporre escalation non è burocrazia quando l'azione produce stato critico. Rifiutare un output non è cattiva esperienza utente quando il dato non può uscire.

La produttività senza blocchi corretti è rischio automatizzato.

Il criterio minimo: il sistema deve trattare arresto, richiesta di prova, riduzione del permesso ed escalation come risultati attesi, osservabili e misurabili.

Se l'unico successo è "ha fatto quello che gli è stato chiesto", l'agente è obbediente.

Non è governato.

6. L'audit guarda le azioni, non i riassunti

Il report finale dell'agente non è una prova.

Può essere utile. Può spiegare. Può aiutare una persona a capire. Ma non basta per dimostrare governance.

Un audit serio deve vedere la catena delle decisioni: input, dati toccati, strumenti chiamati, mandato invocato, policy valutate, autorizzazioni concesse o negate, prove prodotte, output generati, vincoli ereditati, eccezioni aperte, revoche applicate.

Il criterio minimo: ogni azione rilevante deve lasciare una traccia verificabile indipendente dal racconto dell'agente.

Se chiedi all'agente "spiegami perché andava bene" e quella è la tua evidenza principale, non hai audit.

Hai una testimonianza interessata.

7. Il controllore va governato

Quando gli agenti lavorano a velocità macchina, anche il controllo deve accelerare. Ma un controllo dinamico non è neutro solo perché è tecnico.

Decide soglie. Applica attrito. Blocca lavoro. Concede eccezioni. Interpreta contesto. Usa modelli del mondo aziendale. Distingue normale da anomalo. A volte giudica altri modelli. A volte ferma persone e processi prima che qualcuno abbia il tempo di capire cosa sta succedendo.

Questo potere va governato.

Il criterio minimo: il control plane degli agenti deve essere spiegabile, verificabile, correggibile e limitato.

Spiegabile, per permettere contestazioni reali.

Verificabile, per superare revisione tecnica e normativa.

Correggibile, per evitare che ogni falso positivo diventi un'eccezione permanente.

Limitato, per impedire al ragionamento dinamico di riscrivere da solo i principi di sicurezza.

Il controllore non può diventare una scatola nera più potente dell'agente che dovrebbe limitare.

La soglia minima

Questi sette principi non rendono semplice l'adozione degli agenti.

Rendono visibile il costo reale.

Se un progetto agentico enterprise non sa rispondere a queste domande, non è necessariamente da buttare. Ma non è pronto per agire su processi critici.

Dove vive la policy che blocca?

Quale autorità accompagna l'azione?

La delega restringe o allarga?

I vincoli seguono dati, sintesi e output?

Il sistema sa fermarsi?

L'audit vede azioni o solo racconti?

Chi controlla il controllore?

Sono domande scomode, quindi utili.

Gli agenti AI entreranno comunque nelle imprese. Alcuni arriveranno dalla porta principale, con architetture dichiarate, responsabilità, budget e revisione. Altri entreranno di lato, dentro strumenti già comprati, automazioni locali, integrazioni provvisorie, esperimenti diventati produzione per stanchezza.

La differenza tra maturità e improvvisazione non sarà dire "abbiamo agenti".

Sarà poter dimostrare che quando agiscono, qualcuno ha progettato il confine entro cui possono farlo.

Un agente AI non è una chat con le mani.

E se ha le mani nei sistemi aziendali, la sicurezza non può limitarsi a guardarle muoversi.

Appendice

Standard operativi minimi

Questa appendice non vuole trasformare l'ebook in un manuale di prodotto. Sarebbe il modo più rapido per renderlo vecchio prima ancora di distribuirlo.

Serve a fare un'altra cosa: abbassare la tesi architetturale fino al punto in cui un team può usarla per discutere un'implementazione reale senza rifugiarsi nelle parole comode. Governance, sicurezza, controllo, escalation, audit. Tutte parole dignitose, finché qualcuno non chiede dove passano i millisecondi, chi aspetta l'umano, cosa resta bloccato, quale dato può contaminare un comando.

Gli agenti enterprise non falliscono solo perché manca una grande teoria. Falliscono perché una teoria giusta viene collegata male a sistemi che hanno code, timeout, cache, ticket aperti, ruoli sporchi e persone che devono comunque lavorare.

La soglia minima operativa è questa: ogni controllo deve sapere se è bloccante, differibile o osservativo. Ogni escalation deve sapere che cosa congela e per quanto tempo. Ogni input non fidato deve restare separato dal canale che costruisce il comando.

Controlli bloccanti e controlli differibili

Non tutti i controlli meritano lo stesso posto nel flusso.

Alcuni devono stare in linea, anche se costano latenza. Sono quelli che, se arrivano dopo, arrivano inutili: autorizzazione del tool, verifica del mandato, classificazione minima del dato, presenza della prova obbligatoria, blocco su azioni irreversibili, divieto di esportazione, separazione tra ruoli incompatibili.

Se un agente sta per inviare una comunicazione esterna, modificare uno stato contabile, aprire accesso a un sistema, esportare dati cliente o attivare una procedura con effetto reale, il controllo deve precedere l'azione. Non perché sia elegante. Perché dopo non è più controllo. È archeologia.

Altri controlli possono essere differiti, arricchiti o eseguiti in streaming: anomaly detection, scoring comportamentale, revisione qualità, analisi dei pattern di costo, controllo statistico dei falsi positivi, suggerimenti di tuning, correlazione con incidenti precedenti. Sono importanti, ma non devono fingere di essere cancelli se arrivano quando il treno è già partito.

La distinzione pratica è brutale: un controllo bloccante deve poter dire no prima dell'effetto. Un controllo differibile deve poter migliorare il sistema dopo l'effetto. Mescolarli produce architetture lente dove non serve e permissive dove serviva attrito.

Il budget della latenza

Un control plane agentico ha un costo. LLM inference, validazione dello schema, valutazione policy, token exchange, controllo identità, chiamata al tool, logging verificabile. In un flusso multi-agente questo costo si moltiplica, perché ogni delega può introdurre un nuovo punto di decisione.

Dire "mettiamo un policy engine esterno" è corretto, ma incompleto. La domanda successiva è: quante volte lo chiamiamo, con quale granularità e con quale budget?

Una scelta ragionevole è separare tre classi.

- **Decisioni dure:** poche, sincrone, non aggirabili. Azioni critiche, dati sensibili, cambi di stato irreversibili.
- **Decisioni cacheabili:** autorizzazioni brevi, legate a mandato, scope e TTL ridotto. Non "questo agente può fare X per sempre", ma "questo agente può fare questa famiglia di azioni per questo compito, per pochi minuti, con questi limiti".
- **Decisioni osservabili:** valutazioni utili a migliorare controllo, audit e comportamento, ma non necessarie per bloccare ogni singolo passo.

La cache è accettabile solo se riduce latenza senza trasformarsi in diritto permanente. Deve avere scadenza breve, revoca chiara, vincoli ereditati e tracciabilità. Se non sai spiegare perché una decisione cacheata era ancora valida al momento dell'azione, non hai ottimizzato. Hai solo nascosto il rischio dietro una performance migliore.

Human-in-the-loop senza teatro

"Chiama l'umano" sembra prudente. Spesso è solo un modo educato per spostare il problema.

L'essere umano non vive alla velocità dell'agente. Risponde in minuti, ore, a volte mai. Se ogni dubbio diventa escalation, il sistema produce fatica da approvazione. Dopo un po' l'operatore non controlla più: clicca. E quando l'approvazione umana diventa riflesso, la sicurezza ha solo aggiunto un passaggio lento a una decisione automatica.

Un HITL serio deve definire almeno cinque cose.

- **Snapshot:** quale stato dell'agente viene congelato al momento dell'escalation.
- **Ambito sospeso:** quale azione resta bloccata e quali parti del lavoro possono continuare senza aumentare il rischio.
- **Timeout:** quanto dura l'attesa prima che il mandato scada, venga ridotto o torni in coda.
- **Rollback:** quali modifiche parziali devono essere annullate, compensate o marcate come incomplete.
- **Responsabilità:** chi approva cosa, con quali informazioni minime e con quale evidenza registrata.

L'interfaccia di approvazione non deve chiedere all'umano di rileggere tutta la storia dell'agente. Deve mostrare il differenziale: azione richiesta, rischio, policy coinvolta, prova disponibile, conseguenza del sì, conseguenza del no, scadenza dell'approvazione.

Se l'umano deve fidarsi del riassunto prodotto dallo stesso agente che chiede permesso, siamo tornati all'audit circolare. L'approvazione deve poggiare su dati verificabili, non su una narrazione convincente.

Stato, sospensione e ripresa

Un agente sospeso non è una pausa romantica nel flusso. È uno stato distribuito.

Ha letto dati. Ha prodotto ragionamenti intermedi. Forse ha creato bozze. Forse ha chiamato tool non critici. Forse ha aperto sotto-task. Forse ha passato contesto ad altri agenti. Quando arriva l'escalation, il sistema deve sapere cosa resta valido e cosa diventa sospetto.

La ripresa non dovrebbe essere "continua da dove eri rimasto" in modo generico. Dovrebbe essere una nuova decisione vincolata: riprendi questo mandato, con questa approvazione, entro questa finestra, senza usare dati revocati, senza riaprire tool scaduti, registrando che lo stato è stato sospeso e poi riattivato.

Il punto non è complicare tutto. Il punto è evitare che l'HITL diventi una porta laterale attraverso cui uno stato vecchio rientra come se fosse ancora fresco.

Separazione tra dati e comandi

La separazione tra canale dati e canale di controllo va implementata, non proclamata.

Il materiale letto dall'agente deve essere trattato come input non fidato anche quando proviene da fonti aziendali. Una mail interna può essere compromessa. Un ticket può contenere testo ostile. Un documento RAG può portarsi dietro istruzioni mascherate. Una pagina wiki può essere vecchia, manipolata o semplicemente sbagliata.

Per ridurre il rischio servono confini pratici.

- I parametri dei tool devono essere costruiti da schema e mandato, non da testo libero copiato dal dato.
- Le istruzioni operative devono arrivare da canali separati rispetto ai documenti analizzati.
- Il policy engine deve ricevere fatti tipizzati, provenienza, classificazione e richiesta d'azione, non un prompt narrativo.
- Gli output verso sistemi esterni devono passare da allowlist di campi e destinazioni.
- Il recupero RAG deve conservare provenienza, classificazione e vincoli di uso, non solo contenuto testuale.

L'agente può usare i dati per proporre. Il sistema deve usare strutture verificabili per autorizzare. Questa è la differenza tra un assistente che legge documenti e un processo che lascia ai documenti il potere di cambiare le regole.

La checklist prima della produzione

Prima di portare un agente su processi reali, il team dovrebbe poter rispondere a queste domande senza slide motivazionali.

Principio	Domanda di controllo	Criterio di blocco
Enforcement	Dove sono i controlli bloccanti?	Nessuna azione critica va in produzione se il blocco arriva dopo l'effetto.
Latenza	Qual è il budget massimo per una decisione critica?	Se il budget non è dichiarato, il controllo verrà aggirato o disattivato.
Cache / JIT	Quali autorizzazioni sono cacheabili, con quale TTL e quale revoca?	Cache senza scadenza breve e revoca verificabile equivale a permesso permanente.
HITL	Che cosa succede se l'umano non risponde?	Escalation senza timeout e responsabilità produce coda, non controllo.
Stato	Quale stato viene congelato durante escalation?	Se lo stato non è snapshotabile, la ripresa non è auditabile.
Rollback	Quali azioni parziali sono compensabili o irreversibili?	Azione irreversibile senza prova preventiva: no-go.
Data/control separation	Il dato letto dall'agente può influenzare la struttura del comando?	Se sì, il canale dati contamina il canale di controllo.
Policy input	Il policy engine riceve fatti verificabili o testo interpretato?	Testo libero come input normativo: no-go per processi critici.
Supervisione	Chi misura falsi positivi, falsi negativi, costo e attrito operativo?	Nessun owner delle soglie significa nessuna governance del controllore.

Se queste risposte non ci sono, l'agente può anche funzionare in demo. Ma non è pronto per un processo enterprise critico.

Il punto non è bloccare l'adozione. È impedire che la prima architettura seria venga progettata dopo il primo incidente serio.

Fonti

Policy, istruzioni e tool

Liudas Panavas, Sebastian Minus, Bradley Monton, Derek Ray, Suhaas Garre, Sushant Mehta, Edwin Chen, `HANDBOOK.md: A Benchmark for Long-Context Agentic Instruction Following`, arXiv, 2026. https://arxiv.org/abs/2607.25398v1

Repository benchmark `HANDBOOK.md`. https://github.com/surge-ai/handbook

Anthropic, `Introducing the Model Context Protocol`, 2024. https://www.anthropic.com/news/model-context-protocol

OpenHands, progetto open source. https://github.com/All-Hands-AI/OpenHands

Zero Trust e Beyond Zero

Rory Ward, Betsy Beyer, `BeyondCorp: A New Approach to Enterprise Security`, USENIX ;login:, dicembre 2014. https://www.usenix.org/publications/login/dec14/ward

Joseph Valente, Michal Zalewski, `Beyond Zero: Enterprise Security for the AI Era`, arXiv / ACM Queue, 2026. https://arxiv.org/abs/2605.22985

DOI del paper `Beyond Zero`. https://doi.org/10.1145/3819083

Heather Adkins, Archana Ramamoorthy, `Going Beyond Zero: A New Paradigm For Enterprise Security`, Google Security Blog, 27 luglio 2026. https://blog.google/security/going-beyond-zero-a-new-paradigm-for-enterprise-security/

Identità, delega e autorizzazione

NIST, `AI Agent Standards Initiative`, pagina ufficiale aggiornata il 20 aprile 2026. https://www.nist.gov/artificial-intelligence/ai-agent-standards-initiative

Takumi Otsuka, Kentaroh Toyoda, Alex Leung, `AI Identity: Standards, Gaps, and Research Directions for AI Agents`, arXiv, 2026. https://arxiv.org/abs/2604.23280

Krti Tallam, `Authorization Propagation in Multi-Agent AI Systems: Identity Governance as Infrastructure`, arXiv, 2026. https://arxiv.org/abs/2605.05440

Sai Varun Kodathala, `aiAuthZ: Off-Host, Identity-Bound Authorization for AI Agents`, arXiv, 2026. https://arxiv.org/abs/2607.05518

M. Llambi-Morillas, D. Fernandez-Fernandez, `Toward cryptographically verifiable authorization for autonomous AI agents`, arXiv, 2026. https://arxiv.org/abs/2607.21325

Supervisione e sicurezza agentica

Google DeepMind, `Securing internal systems against increasingly capable and imperfectly aligned AI`, 2026. https://deepmind.google/blog/securing-the-future-of-ai-agents/

Anshuman Chhabra, Shrestha Datta, Shahriar Kabir Nahin, Prasant Mohapatra, `Agentic AI Security: Threats, Defenses, Evaluation, and Open Challenges`, arXiv / IEEE Access, 2026. https://arxiv.org/abs/2510.23883

Juhee Kim, Xiaoyuan Liu, Zhun Wang, Shi Qiu, Bo Li, Wenbo Guo, Dawn Song, `The Attack and Defense Landscape of Agentic AI: A Comprehensive Survey`, arXiv, 2026. https://arxiv.org/abs/2603.11088

Nota operativa
Se state progettando agenti AI dentro processi reali, la domanda utile non è "quale modello usiamo?". È "quale autorità gli stiamo consegnando, e chi può dimostrarlo quando qualcosa rompe?". Defilippo srls lavora su integrazioni enterprise, piattaforme cloud industriali, sicurezza e architetture ad alta scalabilità.