Cycademy.

TRAIN . COACH . GUIDE .

WHITEPAPER

AGENTIC AI SECURITY PAPERS · NR. 01

Von Model Security zu Agentic Action Security.

Warum autonome KI eine neue Sicherheitsgrenze im Unternehmen braucht – und warum diese Grenze nicht im Modell liegt, sondern an der Aktion.

FÜR

CISO, CIO, CTO, Security Architecture, AI Governance

UMFANG

33 Seiten · Lesezeit ~18 Minuten

STAND

Version 1.0 · August 2026

KURZFASSUNG

Generative KI hat Unternehmen mit einer neuen Art von **Ausgabe** konfrontiert. Die Sicherheitsdiskussion der vergangenen Jahre hat sich entsprechend auf Modelle, Prompts, Kontextdaten und die Frage konzentriert, welche Inhalte ein Modell erzeugen darf.

Mit Agentic AI verschiebt sich das Problem. Ein Agent leitet aus einer Modellausgabe einen nächsten Schritt ab, wählt ein Werkzeug, ruft eine Schnittstelle auf – und verändert damit den Zustand eines realen Systems. Er versendet eine E-Mail, sperrt ein Konto, ändert eine Cloud-Ressource, überträgt Daten an einen externen Dienst.

Damit reicht es nicht mehr, unerwünschte Ausgaben zu verhindern. Unternehmen müssen kontrollieren, **welche Aktionen ein probabilistisches System tatsächlich ausführen darf** – und welche Wirkung diese Aktionen in Summe entfalten dürfen. Wir verwenden in dieser Reihe dafür den Begriff **Action Security**:

DEFINITION

Action Security bezeichnet die Disziplin, die bestimmt und technisch erzwingt, welche extern wirksamen Aktionen ein agentisches System unter welchem Principal, welcher Delegation, welchem Kontext und welchem Schadensradius ausführen darf.

Für folgenreiche Aktionen heißt das: eindeutige Identität, begrenzte Delegation, minimale Fähigkeiten, kontextabhängige Autorisierung **außerhalb des Modells**, nicht umgehbare Vermittlung, kontrollierte Ausführung, vollständige Nachvollziehbarkeit und ein begrenzter kumulativer Schadensradius.

Das Papier beschreibt außerdem, wo dieses Modell endet. Eine Policy vor dem Tool-Aufruf entscheidet nur so präzise, wie die Architektur ihr Bedeutung bereitstellt – und sie sieht immer nur eine Aktion, während der Angriff in der Sequenz liegen kann.

INHALT

1 Die falsche Frage

Warum „Welches Modell dürfen wir einsetzen?“ nicht mehr die Sicherheitsfrage ist.

In vielen Unternehmen beginnt die Diskussion über Agentic AI mit der Modellfrage. Darf ein externer Anbieter verwendet werden? Welche Daten dürfen in einen Prompt? Müssen Ausgaben gefiltert werden? Wie verhindert man Prompt Injection?

Das sind notwendige Fragen. Sie sind nur nicht mehr ausreichend.

Für einen Agenten, der ausschließlich Informationen zusammenfasst, bleibt die Sicherheit seiner Ausgabe der wesentliche Teil des Risikos. Sobald derselbe Agent auf Werkzeuge zugreifen kann, ändert sich die Lage:

OHNE HANDLUNGSBEFUGNIS	MIT HANDLUNGSBEFUGNIS
Ein Assistent formuliert eine E-Mail.	**Ein Agent versendet sie.**
Ein Assistent erkennt eine fehlerhafte Cloud-Konfiguration.	**Ein Agent korrigiert sie in der Produktion.**
Ein Copilot empfiehlt, ein Konto zu sperren.	**Ein Response-Agent deaktiviert es.**

Der Unterschied liegt nicht im Modell. Er liegt in der Architektur um das Modell herum.

Zwischen Modellausgabe und realem Systemzustand existiert plötzlich ein ausführbarer Pfad. Genau dieser Pfad ist die neue Sicherheitsgrenze.

2 Was sich technisch ändert

Probabilistische Entscheidung, autoritative Wirkung.

Ein Sprachmodell nimmt Kontext entgegen und erzeugt eine Ausgabe. Ein Agent erweitert das um Zustand, Planung, externe Informationen und ausführbare Fähigkeiten – und um eine Rückkopplung.

ABB. 1 **Die agentische Rückkopplungsschleife.** Der Kreislauf, der ohne Menschen weiterläuft.

Diese Rückkopplung ist der sicherheitstechnisch entscheidende Teil. Das Ergebnis einer Aktion beeinflusst den nächsten Schritt. Ein Agent kann deshalb mehrere Aktionen hintereinander ausführen, ohne dass zwischen jedem Schritt ein Mensch steht.

Der Tool-Aufruf selbst ist dabei gewöhnliche Software. Ein API-Endpunkt bleibt ein API-Endpunkt. Eine Datenbank führt SQL aus. Ein Cloud Control Plane verarbeitet einen Request. Neu ist ausschließlich die Instanz, die auswählt: welches Werkzeug, welche Funktion, welche Parameter, wann, auf Basis welcher vorherigen Ausgabe.

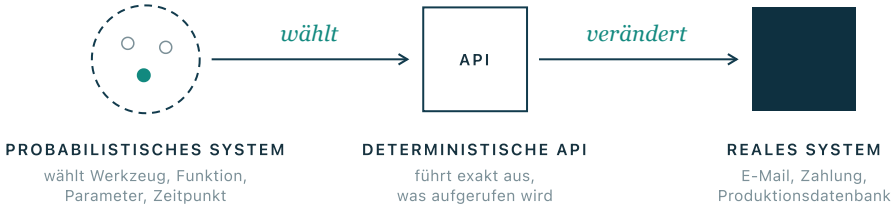

ABB. 2 **Probabilistische Entscheidung, autoritative Wirkung.** Der Tool-Aufruf ist gewöhnliche Software – neu ist die Instanz, die auswählt.

Es liegt nahe, hier von *deterministischer Wirkung* zu sprechen. Das trifft es nicht ganz. Verteilte Systeme antworten mit Wiederholungen, partiellen Fehlern und verzögerter Konsistenz; ein Aufruf kann zweimal ankommen oder nach einem Timeout doch committen. Der sicherheitsrelevante Unterschied ist auch nicht Determinismus. Er ist **Autorität**: Die Ausgabe eines Sprachmodells ist Text. Der Aufruf eines Agenten ist ein Seiteneffekt in einem System, das ihn als legitim behandelt.

Das verändert in jedem Fall die Konsequenz eines Fehlers. In mehrstufigen Workflows kann zusätzlich die Wahrscheinlichkeit einer schädlichen Gesamtabweichung steigen, denn Fehler pflanzen sich fort: Das Ergebnis eines falschen Schritts wird zum Kontext des nächsten.

Ein halluzinierter Servername in einer Empfehlung ist ein Qualitätsproblem. Derselbe halluzinierte Servername als Parameter eines Isolationsbefehls ist ein Produktionsausfall.

Damit entsteht eine Grenze, die Model Guardrails allein nicht absichern können. Ein Systemprompt kann sagen: *Lösche niemals produktive Daten.* Eine Policy Engine kann feststellen:

POLICY-ENTSCHEIDUNG AM ENFORCEMENT POINT

```
action      = DELETE
environment = PRODUCTION
resource    = CUSTOMER_DATABASE
agent       = support-agent
→ DENY
```

Der Prompt formuliert gewünschtes Verhalten. Die Policy erzwingt eine Grenze. Ein Unternehmen würde einen kritischen API-Endpunkt normalerweise nicht dadurch absichern, dass es den aufrufenden Client höflich anweist, bestimmte Operationen zu unterlassen. Bei einem Agenten sollte derselbe Grundsatz gelten.

Ein Modell darf an einer Sicherheitsentscheidung beteiligt sein. Es sollte bei folgenreichen Aktionen nicht die einzige Instanz sein, die entscheidet.

Was „folgenreich“ bedeutet

Der Begriff sollte nicht auf Zustandsänderungen verengt werden. Ein Lesezugriff ändert zunächst nichts – und kann trotzdem der schwerwiegendste Schritt eines Angriffs sein, sobald der gelesene Inhalt in Speicher, Kontext oder einen Ausgangskanal gelangt.

Folgenreich ist eine Aktion, die Sicherheits-, Daten-, Geschäfts- oder Außenwirkung entfalten kann. Dazu gehören neben Schreiben, Löschen und Ausführen ausdrücklich auch: Lesen vertraulicher Daten, Rechte vergeben, Werte übertragen, veröffentlichen, nach außen kommunizieren, verpflichten und freigeben.

Und die Wirkung endet nicht an externen Schnittstellen. Schreibt ein Agent aus einem manipulierten Dokument einen vermeintlichen Fakt in sein Langzeitgedächtnis – etwa, dass ein bestimmter Lieferant Eilzahlungen freigeben darf –, dann ist zunächst nichts passiert. Drei Wochen später, in einem anderen Workflow, wird daraus die Grundlage einer Zahlung. **Persistente Änderungen an Memory, Kontextspeichern, Tool-Katalogen oder Agentenkonfigurationen beeinflussen künftige Entscheidungen und sind deshalb je nach Wirkung selbst folgenreiche Aktionen.** Der Angriff wirkt zeitversetzt, und genau das macht ihn schwer zuzuordnen.

3 Ein realistischer Angriffspfad

Ein Einkaufsagent, ein PDF, und eine Aktion, die niemand beauftragt hat.

Ein Einkaufsagent hat eine überschaubare Aufgabe: Angebote einholen, Dokumente analysieren, Preise vergleichen, Ergebnisse im Beschaffungssystem dokumentieren. Dafür besitzt er Zugriff auf E-Mail, Dokumentenablage, Lieferantendatenbank, Beschaffungssystem und Web.

Einer der Lieferanten schickt ein PDF. Neben dem sichtbaren Angebot enthält es eine Anweisung, die der Agent als Instruktion interpretieren kann.

ABB. 3 **Ein realistischer Angriffspfad.** Entscheidend ist nicht, dass ein Dokument manipulieren kann – sondern dass derselbe Agent auch handeln darf.

Der entscheidende Punkt ist nicht, dass ein Dokument einen Agenten manipulieren kann. Der entscheidende Punkt ist, dass **derselbe Agent auch handeln darf**.

Es gibt zwei grundverschiedene Verteidigungsansätze.
Der erste versucht, die Manipulation zu verhindern – Prompt-Härtung, Content-Prüfung, Isolierung nicht vertrauenswürdiger Inhalte. Das ist sinnvoll und bleibt notwendig.

Der zweite geht davon aus, dass die Manipulation gelingt.

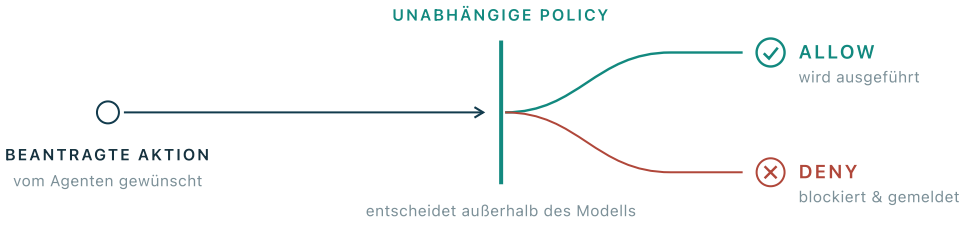

ABB. 4 **Die unabhängige Policy.** Der Prompt formuliert gewünschtes Verhalten – die Policy erzwingt eine Grenze.

Der zweite Ansatz ist unverzichtbar, weil zuverlässige Erkennung natürlicher, kontextabhängiger Manipulation kein realistischer Architekturbaustein ist. Man kann eine Erkennungsrate verbessern. Man kann sie nicht als letzte Sicherheitsgrenze einplanen.

4 Die Kombination, auf die es ankommt

Drei Eigenschaften, die einzeln unproblematisch und zusammen gefährlich sind.

Nicht jeder manipulierbare Agent ist gefährlich. Gefährlich wird ein Agent, der drei Eigenschaften gleichzeitig besitzt – von Simon Willison als *lethal trifecta* beschrieben:

01	**Zugriff auf vertrauliche Daten** – Kundendaten, Quellcode, Verträge, Zugangsdaten.
02	**Verarbeitung von Inhalten, die ein Angreifer beeinflussen kann** – E-Mails, PDFs, Webseiten, Ticket-Kommentare, Tool-Ausgaben, fremde Repositories.
03	**Ein Kanal nach außen** – die Möglichkeit, Informationen aus der Umgebung herauszubringen.

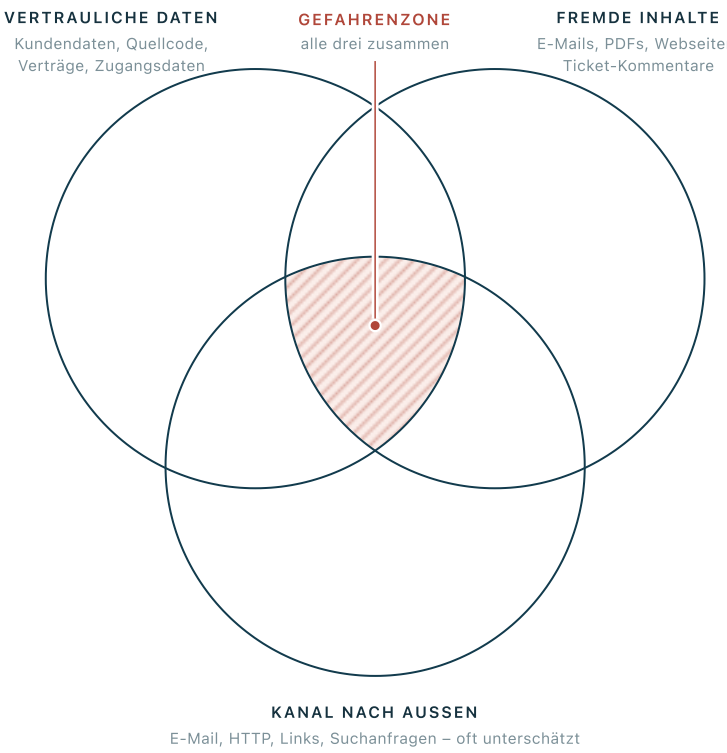

ABB. 5 **Die lethal trifecta.** Einzeln alltäglich, zusammen ein vollständiger Exfiltrationspfad. · nach Simon Willison, 2025

Jede dieser Eigenschaften ist für sich genommen alltäglich. Zusammen ergeben sie einen vollständigen Exfiltrationspfad, den ein Angreifer allein über Inhalte auslösen kann, ohne jemals ein System zu kompromittieren.

Der dritte Punkt wird dabei regelmäßig unterschätzt. „Kanal nach außen" bedeutet nicht nur `send_email()` oder `http_post()`. Es genügt oft erheblich weniger: eine Bildreferenz in einer gerenderten Antwort, deren URL Daten im Pfad transportiert. Ein Web-Search-Aufruf, dessen Suchbegriff frei gewählt werden kann. Ein Namensauflösungsvorgang. Ein Link, den ein Mensch anschließend anklickt. Wer den Ausgangskanal begrenzen will, muss ihn zuerst vollständig kennen – und die vollständige Liste ist fast immer länger als die Liste der Tools.

Die Regel gehört an den Graphen, nicht an den Agenten

Die naheliegende Konsequenz ist eine Zerlegung: ein Agent verarbeitet fremde Inhalte ohne Zugriff auf vertrauliche Daten, ein zweiter arbeitet auf vertraulichen Daten, sieht aber keine fremden Inhalte.

Das löst das Problem nur, wenn die Grenze zwischen beiden hält. Reicht Agent A eine Nachricht an Agent B weiter, und darf B vertrauliche Daten lesen und nach außen kommunizieren, ist die Trifecta über den Verbund wiederhergestellt – verteilt auf zwei Systeme, die einzeln jeweils unauffällig aussehen. In Multi-Agent-Architekturen ist das der Normalfall, nicht die Ausnahme.

Die Entwurfsregel muss deshalb schärfer formuliert werden:

Kein erreichbarer Pfad durch Agenten und Werkzeuge darf die drei Eigenschaften vereinen, ohne dass mindestens eine unabhängige technische Grenze den Daten- oder Aktionsfluss unterbricht.

Das ist eine Aussage über den gesamten erreichbaren Daten- und Aktionsgraphen, nicht über einzelne Komponenten. Sie lässt sich nur beantworten, wenn man diesen Graphen kennt – was direkt zurück auf die Inventarfrage führt.

Herkunft im Kontext – und was sie nicht bedeutet

Zwischen „Manipulation erkennen“ und „Aktion per Policy blockieren“ liegt eine dritte Ebene, die derzeit aus der Forschung in die Praxis wandert.

Herkunftsverfolgung. Jeder Inhalt im Agentenkontext trägt eine Markierung, woher er stammt. Aktionen können dann von der Herkunft der Daten abhängig gemacht werden, die sie ausgelöst haben.
Dabei genügt ein einzelnes Kennzeichen wie „geprüft“ nicht – es vermischt mindestens vier verschiedene Aussagen. Nützlich wird Herkunft erst als strukturiertes Attributbündel:

HERKUNFT ALS STRUKTURIERTES ATTRIBUTBÜNDEL

```
source.type            = supplier_document
source.identity        = supplier-4711
source.authenticated   = true
source.trust_level     = external_untrusted
content.classification = commercial_confidential
instruction_authority  = none
```

Die letzte Zeile ist die wichtigste.

Authentizität ist nicht Autorität. Ein kryptografisch signiertes Lieferantendokument kann zweifelsfrei echt sein – **und trotzdem keinerlei Instruktionsautorität über den Agenten besitzen.**

Trennung von Kontroll- und Datenfluss. Ein privilegiertes Modell plant, ohne fremde Inhalte je zu sehen; ein zweites, isoliertes Modell verarbeitet die fremden Inhalte, kann aber keine Aktionen auslösen. Google DeepMinds *CaMeL*-Arbeit und das ältere Dual-LLM-Muster gehören in diese Familie.

Beides ist noch nicht ausgereift, und beides erfordert Eingriffe in die Agentenarchitektur, die man nachträglich nur schwer macht. Genau deshalb gehört die Frage in die Architekturentscheidung und nicht in die Betriebsphase.

5 Die Sicherheitsgrenze verschieben

Der Agent darf vorschlagen. Über die Zulässigkeit entscheidet er nicht selbst.

Eine tragfähige Grundarchitektur sieht nicht so aus: `User → Agent → API`. Sie sieht so aus:

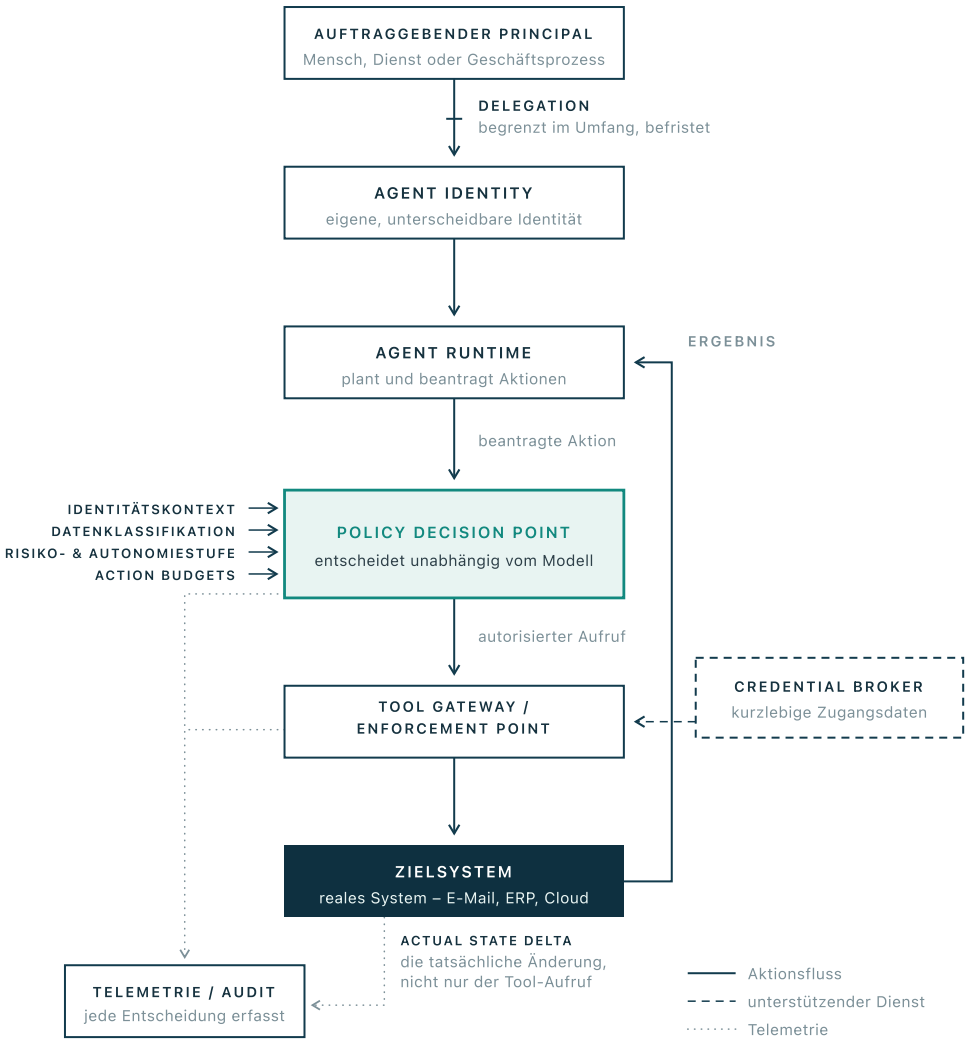

ABB. 6 **Referenzarchitektur.** Die Entscheidung fällt außerhalb der Runtime; das Zielsystem meldet die tatsächliche Änderung an die Telemetrie.

Das zugrundeliegende Prinzip ist nicht neu: **Entscheidung und Durchsetzung liegen außerhalb der Instanz, deren Verhalten kontrolliert werden soll.** Neu ist, dass die kontrollierte Instanz ihre Aktionen nicht mehr aus vorab festgelegter Programmlogik ableitet.

Dabei sind zwei Funktionen zu trennen, die in Agentenprojekten fast immer zu einer verschmelzen. Der **Policy Decision Point** wertet Identität, Delegation, Ressourcenkontext, Datenklassifikation, Herkunft und Budgets aus und *entscheidet*. Der **Policy Enforcement Point** – hier der Tool Gateway – *erzwingt* diese Entscheidung und ist der einzige Weg zum Zielsystem.

Vier Fragen, die oft zu einer verschmelzen

In vielen Agentenprojekten wird „Identität“ als ein einziges Problem behandelt. Tatsächlich sind es vier, und sie brauchen unterschiedliche Mechanismen:

FRAGE	MECHANISMUS
Wer bist du?	**Agent- und Workload-Identität**
Für wen handelst du?	**Delegation**
Was darfst du?	**Autorisierung**
Mit welcher technischen Vollmacht?	**Credential / Capability**

Zwei weitere Unterscheidungen kommen hinzu. Erstens handelt nicht jeder Agent im Auftrag eines Menschen: Ein Response-Agent im SOC wird von einem Alarm ausgelöst, ein Abstimmungsagent von einem Zeitplan, ein nachgelagerter Agent von einem anderen Agenten. Davon zu trennen ist die Frage der Verantwortung – der Benutzer, der einen Workflow startet, ist nicht automatisch derjenige, der geschäftlich oder regulatorisch für das Systemrisiko einsteht. **Auftraggebender Principal und verantwortlicher Owner sind zwei verschiedene Rollen**, und beide gehören ins Inventar.

Zweitens eine Unterscheidung, die für Forensik entscheidend ist: `procurement-agent` **ist nicht dasselbe wie die laufende Instanz dieses Agenten** – in Version 3.4, in einem bestimmten Pod, mit einer bestimmten Modellversion und einem bestimmten Policy-Bundle. Für den Betrieb genügt oft die logische Identität. Für die Rekonstruktion eines Vorfalls nicht.

Delegation ist kein neues Protokollproblem – die Integration ist es

Der häufigste Fehler in produktiven Agenten ist, dass der Agent unter dem Token seines menschlichen Auftraggebers handelt. Damit ist im Nachhinein nicht unterscheidbar, ob der Mensch oder der Agent gehandelt hat, und der Agent erbt sämtliche Rechte des Menschen.
Für die Darstellung dieser Beziehung existiert ein Standard. **RFC 8693 (OAuth 2.0 Token Exchange)** kennt mit dem `act`-Claim die Konstruktion „X handelt im Auftrag von Y“:

RFC 8693 · TOKEN MIT ACT-CLAIM

```
{
  "iss":   "https://id.example.com",
  "sub":   "marcel@example.com",
  "act":   { "sub": "agent://procurement-agent/prod-3" },
  "aud":   "procurement-api",
  "scope": "read:offers write:case_note",
  "exp":   1786118400
}
```

Die Leserichtung ist wichtig: `sub` **benennt den Menschen, in dessen Auftrag gehandelt wird;** `act` **den Agenten, der tatsächlich handelt.** Bei mehrstufiger Delegation ist der äußerste `act` der aktuelle Actor, tiefer geschachtelte sind frühere. RFC 8693 legt außerdem fest, dass ein Token-Konsument für seine Zugriffsentscheidung ausschließlich die Top-Level-Claims und den durch `act` benannten aktuellen Actor heranziehen darf.

Ein zweiter, oft übersehener Claim des RFC ist `may_act`: Er benennt, wer überhaupt berechtigt ist, für ein Subjekt als Actor aufzutreten. Damit lässt sich die Delegations*berechtigung* ausdrücken – nicht nur die Delegations*tatsache*. Ein Claim bleibt allerdings zunächst eine Behauptung: Sicherheit entsteht erst, wenn der Authorization Server diese Beziehung bei der Ausstellung tatsächlich prüft und durchsetzt.

Was RFC 8693 dagegen **nicht** leistet: Aufgaben-, Zweck- und Transaktionsbindung. Der Standard definiert `act`, `may_act`, `scope` und `client_id`. Eine Bindung an einen konkreten Vorgang lässt sich über organisationsspezifische Claims modellieren oder über feingranulare Autorisierungsmechanismen wie **RFC 9396 (Rich Authorization Requests)** mit `authorization_details`.

Und das Protokoll allein löst ohnehin nur einen Teil. Offen bleiben Workload-Identität und Attestierung der Laufzeit, die Frage, wer delegieren *darf*, das Mapping zwischen geschäftlicher Autorität und technischem Principal, Credential Binding, Policy-Lebenszyklus, Widerruf und die Governance mehrstufiger Delegation. Das Protokoll ist vorhandene Infrastruktur; die Integration in Agentenarchitekturen ist die eigentliche Arbeit.

Credentials sollten den Agenten möglichst nie erreichen

Kurzlebige Zugangsdaten sind gut. Besser ist, bei folgenreichen Aktionen gar keine wiederverwendbaren Zugangsdaten an den Agenten zu geben.

Nicht: Der Agent erhält ein Token und ruft damit das Zielsystem. Sondern: Das Gateway führt die freigegebene Aktion mit einem eng gefassten Credential aus, das den Agenten nie erreicht. Ein kompromittierter Agent kann ein Credential, das er nie besessen hat, auch nicht außerhalb des vorgesehenen Pfades verwenden.

Wo Tokens den Agenten doch erreichen müssen, sollten sie an den legitimen Sender gebunden sein – etwa über DPoP –, damit ein entwendetes Token außerhalb seines Kontexts wertlos ist. Senderbindung reduziert Replay und Token-Diebstahl außerhalb der legitimen Laufzeit; gegen eine vollständig kompromittierte Runtime, die auch den privaten Schlüssel kontrolliert, hilft sie nicht.

Eine Policy ist konkreter als ein Prompt

Eine Autorisierungsregel für eine folgenreiche Aktion lässt sich deklarativ ausdrücken:

DEKLARATIVE AUTORISIERUNGSREGEL

```
permit (
  principal in AgentGroup::"procurement",
  action    == Action::"create_purchase_order",
  resource  in Environment::"production"
) when {
  context.delegation.human_principal == resource.cost_center_owner &&
  context.amount <= 5000 &&
  context.supplier in Allowlist::"approved_suppliers" &&
  context.source_documents.all(d => d.instruction_authority == "none")
};
```

Die letzte Zeile verbindet die Autorisierung mit der Herkunft der Daten, auf denen die Entscheidung beruht – die Brücke zu Kapitel 4.

Das Ergebnis muss nicht binär sein.

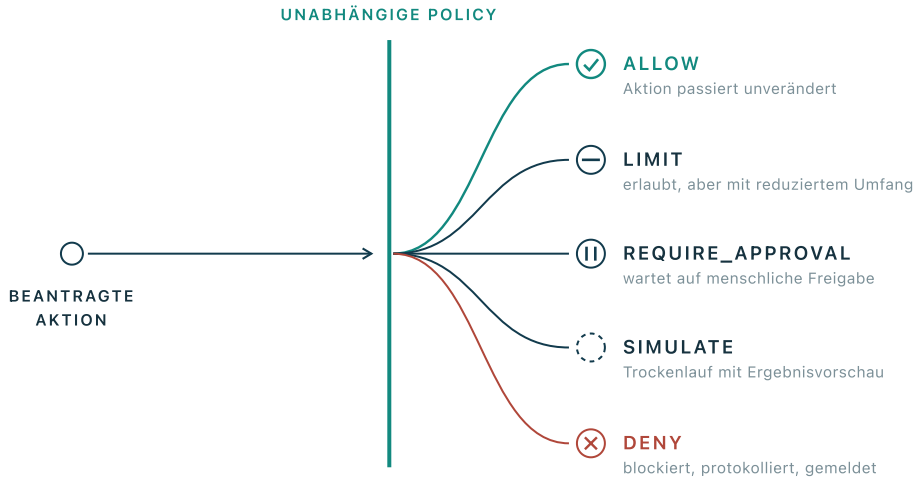

ABB. 7 **Abgestufte Policy-Entscheidungen.** Zwischen Erlauben und Verbieten liegen Begrenzen, Freigeben lassen und Simulieren.

`LIMIT` erlaubt die Aktion mit reduziertem Umfang, `SIMULATE` führt sie als Trockenlauf mit Ergebnisvorschau aus. Beide Ausgänge fehlen in den meisten Implementierungen – und beide sind genau dort nützlich, wo ein hartes `DENY` den Agenten unbrauchbar machen würde.

„Das ist doch nur Zero Trust“

Der Einwand liegt nahe und ist zur Hälfte berechtigt. NIST SP 800-207 verschiebt Vertrauen weg von Netzwerkpositionen hin zu expliziten Prüfungen, kennt dynamische, kontextabhängige Policy Enforcement Points und die laufende Neubewertung von Zugriffen; SP 800-207A ergänzt granulare Policies auf Basis von Anwendungs- und Dienstidentitäten. Diese Prinzipien werden hier angewendet, nicht ersetzt.

Der Unterschied liegt in der Granularität und im Gegenstand der Entscheidung:

> Action Security übernimmt die Prinzipien von Zero Trust – explizite Identität, minimale Autorität, kontextabhängige Entscheidungen, externe Enforcement Points – **und verschiebt deren Granularität von der Ressource und der Sitzung auf die konkrete, agentisch gewählte Aktion und ihren erwarteten Effekt.**

Hinzu kommt eine Eigenschaft, die klassische Architekturen nicht voraussetzen mussten: Der Handelnde ist nicht deterministisch. Zero Trust prüft, *ob* ein Subjekt zugreifen darf. Bei Agenten ist zusätzlich unbestimmt, *was* das Subjekt als Nächstes tun wird – auch ohne Angreifer.

6 Sechs Fragen, sechs Invarianten

Das Skelett der Serie.

Bevor ein Agent eine folgenreiche Aktion ausführt, sollte die Architektur sechs Fragen beantworten können. Jede Frage korrespondiert mit einer Eigenschaft, die das System danach besitzen muss – die Frage ist der Prüfpunkt, die Invariante das Ergebnis. Jede davon ist ein eigenes Arbeitsgebiet; dieses Papier stellt sie auf, die Folgepapiere gehen in die Tiefe.

#	FRAGE	INVARIANTE	VERTIEFT IN
1	Wer handelt?	**IDENTIFIZIERBAR** – jede Aktion hat einen eindeutig bestimmbaren Actor	#03
2	In wessen Auftrag?	**DELEGIERT** – fremde Autorität ist explizit und begrenzt übertragen	#03
3	Welche Fähigkeit steht zur Verfügung?	**MINIMIERT** – der Agent besitzt nur, was seine Aufgabe braucht	#04
4	Ist genau diese Aktion zulässig?	**VERMITTELT** – jede folgenreiche Aktion durchläuft eine nicht umgehbare Grenze	#05
5	Wie wird sie ausgeführt?	**BEGRENZT** – Einzelaktion *und* Workflow haben einen begrenzten Schadensradius	#06
6	Können wir sie nachvollziehen und stoppen?	**REKONSTRUIERBAR** – unabhängig und manipulationsgeschützt belegbar	#08

1 · Wer handelt? Der Agent braucht eine identifizierbare Laufzeitidentität. `user@example.com` ist unzureichend, wenn tatsächlich ein autonomer Agent unter dem Token des Benutzers gehandelt hat. Sichtbar bleiben muss: dass ein Agent gehandelt hat, welche Instanz es war, und auf wessen Veranlassung.

2 · In wessen Auftrag? Identität und Befugnis sind verschiedene Dinge. Eine Delegation beantwortet: wer delegiert, was, an wen, wofür, auf welche Ressource, wie lange – und wer überhaupt delegieren darf. Dass ein Mensch eine Rückerstattung freigeben darf, bedeutet nicht, dass diese Fähigkeit an einen Agenten delegiert werden sollte.

3 · Welche Fähigkeit steht zur Verfügung? Ein Agent kann eine Funktion, die ihm gar nicht zur Verfügung steht, auch nach erfolgreicher Manipulation nicht aufrufen. Least Privilege begrenzt die Rechte einer Identität; **Least Capability** begrenzt den Handlungsraum des Agenten. Ein Agent zur E-Mail-Zusammenfassung braucht kein `send_mail()`.

4 · Ist genau diese Aktion zulässig? Die Runtime-Autorisierungsfrage. Die Antwort hängt vom Kontext ab: Agent, Auftraggeber, Aktion, Ressource, Umgebung, Datenklassifikation, Betrag, Zeitpunkt, Umkehrbarkeit, Herkunft der auslösenden Daten.

5 · Wie wird sie ausgeführt? Eine erlaubte Aktion kann trotzdem Schaden anrichten – und eine Folge erlaubter Aktionen erst recht. Dazu gehören Parametervalidierung, Sandboxing, Egress-Kontrolle, kurzlebige oder gar keine Credentials, Allowlisting der Zielsysteme sowie Budgets für Menge, Wert und Änderungsumfang.

6 · Können wir sie nachvollziehen und stoppen? Nach einem Vorfall muss rekonstruierbar sein: welche Agenteninstanz, unter welcher Delegation, welches Tool, mit welchen Parametern, gegen welche Ressource, mit welcher Policy-Entscheidung – und **welche Zustandsänderung tatsächlich eingetreten ist**. Ein Protokolleintrag „Agent rief `disable_user("alice")` auf, HTTP 200“ belegt nicht, dass Alice gesperrt ist: Aufrufe laufen doppelt, Timeouts committen trotzdem, asynchrone Jobs scheitern später. Dazu Mechanismen zum Eingreifen: Credential Revocation, Agent Isolation, Tool Disablement, Circuit Breaker, Rollback.

Diesen sechs Fragen ist eine siebte vorgelagert, die in den meisten Unternehmen unbeantwortet ist: **Welche Agenten führen bei uns überhaupt bereits Aktionen aus?** → *Paper #02 · Agent Inventory*

7

Was diese Grenze nicht leistet

Sechs Einschränkungen, die man kennen muss, bevor man sich auf das Modell verlässt.

Eine unabhängige Entscheidung vor dem Tool-Aufruf, durchgesetzt an einem Enforcement Point, ist eine wirksame Grenze. Er ist keine vollständige. Sechs Punkte begrenzen ihn systematisch – die letzten beiden sind die, an denen Implementierungen in der Praxis am häufigsten scheitern.

1 · Eine Policy entscheidet nur so präzise, wie ihr Bedeutung bereitgestellt wird

`send_email(to, body)` trägt kaum Semantik. Eine Policy kann den Empfänger gegen eine Allowlist prüfen, aber nicht erkennen, ob der Textkörper vertrauliche Inhalte transportiert. Das ist der klassische *confused deputy*: Der Agent besitzt legitime Befugnisse und wird dazu gebracht, sie für fremde Zwecke einzusetzen. Die Aktion ist erlaubt. Die Absicht dahinter nicht.
Es wäre allerdings falsch zu sagen, eine Policy sehe grundsätzlich nur Syntax. Ein gut angebundener Decision Point kann sehr wohl Bedeutung erhalten: Empfängerdomäne intern oder extern, Datenklassifikation, erwartete Zustandsänderung, Ressourcenkritikalität, DLP-Befund. Die Einschränkung ist deshalb keine Eigenschaft des Konzepts, sondern eine Designregel:

Ein Agent entscheidet dabei auch anhand dessen, was ihm über seine Werkzeuge *gesagt* wird: Name, Beschreibung, Schema, Server-Metadaten. Ein manipulierter Werkzeugkatalog kann ein Modell zu einer falschen Aktion bewegen, obwohl das aufgerufene Werkzeug selbst legitim ist. **Tool-Definitionen sind deshalb sicherheitsrelevante Konfiguration** – authentifiziert, versioniert, änderungskontrolliert und integritätsgeschützt.

Die Qualität einer Autorisierungsentscheidung kann die Qualität des bereitgestellten Kontexts nicht übersteigen. **Eine Policy entscheidet nur so präzise, wie die Architektur die sicherheitsrelevante Bedeutung einer Aktion strukturiert verfügbar macht.**

Daraus folgt unmittelbar ein Architekturprinzip: **Semantisch schmale Werkzeuge sind nicht nur Least Capability – sie sind die Voraussetzung für sinnvolle Autorisierung.** Statt `sql("UPDATE …")` ein `suspend_customer_account(customer_id, duration, reason_code)`. Statt `http_post(url, body)` ein `submit_supplier_quote(supplier_id, quote_id)`. Der Zuschnitt der Werkzeuge ist damit selbst Teil des Sicherheitsmodells.

2 · Der Agent darf die Fakten seiner eigenen Autorisierung nicht selbst festlegen

Eine kontextabhängige Autorisierung ist nur so vertrauenswürdig wie ihre Eingaben. Ein Teil davon muss zwangsläufig vom Agenten kommen – welche Aktion er beantragt, für welchen Lieferanten, über welchen Betrag. Ohne diese Angaben gäbe es nichts zu entscheiden. Die Trennung verläuft deshalb nicht zwischen „vom Agenten“ und „nicht vom Agenten“, sondern zwischen dem, was er *beantragt*, und dem, was er *bezeugt*:

```
BEANTRAGT   vom Agenten, nicht vertrauenswürdig
  requested.amount        = 4.200 EUR
  requested.supplier      = supplier-4711

BEZEUGT     aus autoritativen Quellen
  supplier.status         = approved       (ERP)
  delegation.limit        = 5.000 EUR      (IAM)
  resource.classification = internal       (Datenkatalog)

ENTSCHEIDUNG
  requested.amount <= delegation.limit AND supplier.status == approved
```

Teilt der Agent selbst mit, wie kritisch die betroffenen Daten sind oder wie hoch das Risiko ist, dann kontrolliert ein Angreifer über den manipulierten Kontext die Autorisierungsentscheidung – und die Grenze wird zur Fassade. Datenklassifikation, Umgebung, Ressourcenkritikalität und Delegationsumfang müssen aus dem Token, der CMDB, dem Datenkatalog oder der Registry stammen.

3 · Autorisierungsentscheidungen altern

Ein Plan wird zum Zeitpunkt T_0 freigegeben. Schritt sieben läuft bei T_5 – gegen einen Kontext, der sich verändert hat, mit Parametern aus zwischenzeitlichen Ergebnissen. Bei langlaufenden Workflows ist die Lücke zwischen Prüfung und Ausführung kein theoretisches Problem, sondern der Normalfall. Wer einen Plan freigibt, gibt nicht dieselbe Sache frei wie den einzelnen Schritt. Konsequenz: kurze Gültigkeiten, erneute Prüfung an jeder folgenreichen Aktion, und Freigaben, die an konkrete Parameter gebunden sind statt an eine Absichtserklärung.

4 · Grob geschnittene Fähigkeiten entwerten die Grenze

`shell(command)` ist genau ein Tool-Aufruf und beliebig viele Aktionen. `get_service_status(service_id)` ist eine Aktion mit prüfbaren Parametern. Eine sehr mächtige Fähigkeit kann beim Aufruf sauber autorisiert werden – innerhalb davon findet keine Policy mehr statt.

5 · Erlaubte Einzelaktionen ergeben nicht zwingend einen erlaubten Workflow

Das ist die Lücke, die in den meisten Architekturen offen bleibt. Eine Autorisierungsentscheidung sieht eine Aktion. Der Angriff kann in der Sequenz liegen.

JEDE AKTION ERLAUBT · DER WORKFLOW EINE DATENABSCHÖPFUNG

```
read_customer(000001)  → ALLOW
read_customer(000002)  → ALLOW
read_customer(000003)  → ALLOW
…
read_customer(100000)  → ALLOW
```

Jede einzelne Aktion ist legitim. Der Workflow ist eine Datenabschöpfung.

Dasselbe Muster in anderen Domänen: Jeder einzelne Knoten darf isoliert werden – zusammen fällt der Cluster aus. `create_user`, `grant_role`, `generate_api_key`, `send_email` sehen einzeln unauffällig aus; die Sequenz ist die Einrichtung eines dauerhaften Zugangs.
Eine Architektur, die jede Aktion isoliert autorisiert, kann eine unerlaubte Gesamtwirkung also durchaus zulassen. Agentische Systeme brauchen deshalb neben der Autorisierung auf Aktionsebene auch **Grenzen auf Workflow- und Verlaufsebene**:

BUDGETS AUF VERLAUFSEBENE

```
max_customer_records_per_task     =    50
max_production_changes_per_flow   =     3
max_external_recipients           =     1
max_payment_value_per_delegation  =  5.000 EUR
max_payment_value_per_day         = 10.000 EUR
```

Dazu kommen Rate Limits, Nebenläufigkeitsgrenzen, kumulative Daten- und Änderungsbudgets, Sequenz-Policies und Risikoakkumulation über den Verlauf.

Entscheidend ist dabei, wo das Budget geführt wird. Ein Zahlenwert im Agentenkontext ist keine Grenze: Laufen zwei Instanzen desselben Agenten parallel, sehen beide dasselbe verbleibende Budget und schöpfen es beide aus. **Budgets müssen zentral, atomar und workflowübergreifend verbucht werden** – reservieren, autorisieren, ausführen, festschreiben oder freigeben. Alles andere ist eine Empfehlung, keine Kontrolle.

Damit zerfällt die Frage in zwei Ebenen:

```
Darf diese Aktion stattfinden?              →  Aktionsebene
Darf diese Folge von Aktionen stattfinden?  →  Verlaufsebene
```

6 · Eine Grenze, die man umgehen kann, ist keine

Die Architektur aus Kapitel 5 funktioniert nur, wenn der Agent den Enforcement Point nicht umgehen kann. Besitzt ein Agent neben dem Tool Gateway auch eine unbeschränkte Shell, rohen Netzwerkzugriff, direkte Datenbank-Zugangsdaten oder ein Cloud-CLI, dann ist das Gateway kein Sicherheitsmechanismus, sondern eine Konvention.

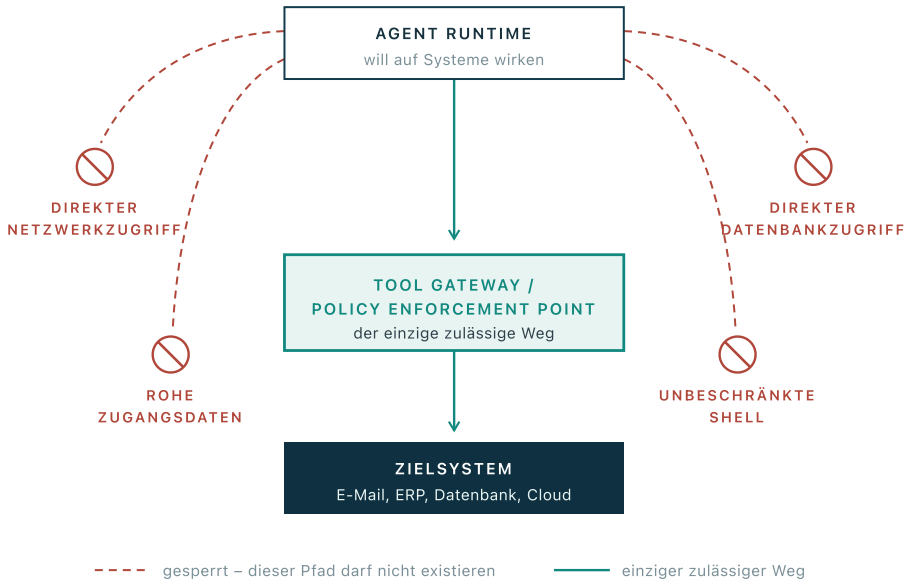

ABB. 8 **Vollständige Vermittlung.** Ein Tool Gateway neben einer offenen Shell ist kein Gateway.

Das ist das klassische Prinzip der **vollständigen Vermittlung**: Jede sicherheitsrelevante Zustandsänderung muss vollständig und nicht umgehbar über die Grenze laufen. Es ist keine neue Erkenntnis – aber es ist die Anforderung, die in Agentenprojekten am häufigsten stillschweigend verletzt wird, weil eine Shell für die Entwicklung eben praktisch ist.

Das hat eine Konsequenz, die man einplanen muss: Wenn jede Agentenaktion durch Entscheidungsdienst, Gateway, Credential Broker und Budgetführung läuft, werden diese Komponenten selbst hochkritisch – Tier-0-Infrastruktur mit allem, was dazugehört: Verfügbarkeit, administrative Trennung, Änderungskontrolle, Policy-Rollback, Break-Glass. Und die Frage, die vor dem ersten Ausfall beantwortet sein muss: **Was passiert, wenn der Entscheidungsdienst nicht antwortet?** Für folgenreiche Aktionen kann die Antwort nur *fail closed* lauten. Nicht vertretbar ist eine Architektur, die diese Frage offenlässt und im Störfall durchlässig wird.

Das Ziel ist nicht, ein probabilistisches System deterministisch zu machen. Das Ziel ist eine Architektur, in der eine falsche Entscheidung nicht automatisch zu einem unkontrollierten Ergebnis führt.

8 Was das für Leitung und Security bedeutet

Risiko steuert Kontrolltiefe – nicht umgekehrt.

Nicht jede Aktion braucht dieselbe Kontrolle

Ein häufiger Fehler wäre, aus Action Security ein universelles Freigabesystem zu machen. Wenn jeder Tool-Aufruf einen Menschen benötigt, verschwindet der Nutzen autonomer Systeme vollständig.

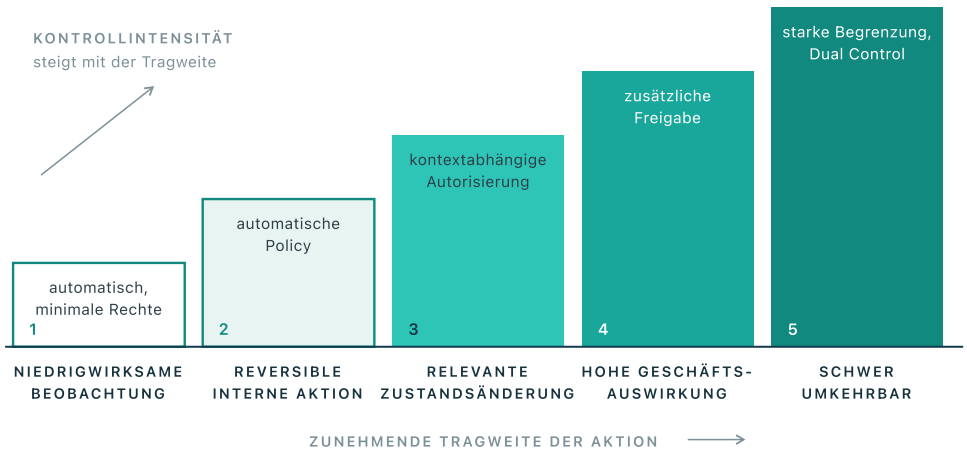

ABB. 9 Aktionsklassen und typische Kontrolle. Maßgeblich ist die geschäftliche Umkehrbarkeit der Wirkung, nicht die technische der Aktion.

Die aufschlussreichste Einzelgröße ist dabei nicht die Berechtigung, sondern die **Umkehrbarkeit** – und zwar die geschäftliche, nicht die technische.

Ein Konto lässt sich entsperren. Der durch die Sperre ausgefallene Zahlungslauf nicht. Eine Firewall-Regel lässt sich entfernen. Der Produktionsstillstand während ihrer Wirkung bleibt real. Veröffentlichte vertrauliche Informationen lassen sich überhaupt nicht zurückholen.

Die Frage lautet deshalb nicht: *Kann die Aktion rückgängig gemacht werden?* Sondern: **Kann ihre Wirkung rechtzeitig und vollständig rückgängig gemacht werden?**

Umkehrbarkeit allein reicht für eine Einstufung allerdings nicht. Eine belastbare Klassifikation berücksichtigt daneben mindestens Schadenshöhe, Reichweite, Außenwirkung, Veränderung von Sicherheitsautorität, Datensensitivität, Autonomiegrad der Folgeschritte und Erkennbarkeit eines Fehlers. Paper #05 operationalisiert diese Dimensionen; für #01 genügt der Grundsatz, dass die Kontrolltiefe aus dem Risiko folgt und nicht aus der Technik.

Menschliche Freigabe: notwendig, aber kein Ersatz für Architektur

Menschliche Freigabe ist ein wertvoller Kontrollmechanismus für seltene, hochriskante, schwer umkehrbare Aktionen. Sie ist keine vollständige Sicherheitsarchitektur.

Ein Agent, der einem Analysten fünfzigmal pro Schicht denselben Dialog zeigt – *Konto sperren? [Freigeben] [Ablehnen]* – erzeugt kein Vier-Augen-Prinzip, sondern eine Bestätigungsschleife. Freigabe, die zur Routine wird, ist Dokumentation, keine Kontrolle.

Wer kontrolliert eigentlich, was der Mensch zu sehen bekommt?

Diese Frage wird fast nie gestellt, und sie ist die gefährlichste in diesem Kapitel. Wenn der Agent dem Menschen mitteilt „Ich möchte Lieferantenangebot #42 bestätigen. Auswirkung: keine“, und genau dieser Agent ist kompromittiert, dann ist die Freigabemaske Teil des Angriffs. Bei folgenreichen Aktionen darf der Freigebende deshalb nicht ausschließlich eine vom Agenten erzeugte Zusammenfassung sehen. Die Maske sollte ihre Angaben aus vertrauenswürdigen Systemen beziehen:

FREIGABEMASKE AUS VERTRAUENSWÜRDIGEN QUELLEN

AKTION	Bestellung anlegen	
LIEFERANT	ACME GmbH	[aus Lieferantenregister verifiziert]
BETRAG	47.250 EUR	
HERKUNFT	extrahiert aus externem PDF	[nicht verifiziert]
BUDGETVERANTWORTUNG	Marcel Example	[aus Kostenstellenstammdaten]
ZUSTANDSÄNDERUNG	neue vertragliche Verpflichtung, 47.250 EUR	
UMKEHRBARKEIT	gering	
POLICY	menschliche Freigabe erforderlich	

Und die Freigabe sollte an genau diese Aktion gebunden sein – an Actor, Ressource, normalisierte Parameter und Policy-Version, mit Ablaufzeit. Ändert der Agent danach den Betrag von 5.000 auf 50.000, ist die Freigabe ungültig. Paper #05 beschreibt die Umsetzung.

Das Ziel ist nicht Human-in-the-Loop überall. Das Ziel ist Automatisierung dort, wo das Risiko begrenzt ist – und menschliche Entscheidung dort, wo menschliches Urteil tatsächlich einen Sicherheitsgewinn erzeugt.

Zur Einordnung gegenüber unserem Papier *NIS2 trifft Agentic AI*: Dort steht, dass eine meldepflichtige Entscheidung nicht an ein probabilistisches Modell delegiert werden darf – der Agent bereitet vor, der Mensch entscheidet. Das gilt unverändert. Es ist die richtige Antwort für eine kleine Zahl regulatorisch gebundener, ohnehin seltener Entscheidungen. Action Security beantwortet die andere Hälfte: was mit den tausend Aktionen pro Tag geschieht, bei denen eine menschliche Freigabe weder vorgeschrieben noch durchhaltbar ist.

Der regulatorische Anschluss

Für Unternehmen im Anwendungsbereich europäischer Regulierung zeigt eine Reihe von Anforderungen in dieselbe Richtung.

Der **EU AI Act** verlangt, dass Hochrisiko-KI-Systeme so konzipiert werden, dass sie während ihrer Nutzung wirksam durch natürliche Personen beaufsichtigt werden können; die Maßnahmen müssen dem Risiko, dem Autonomiegrad und dem Nutzungskontext angemessen sein (Art. 14). Daraus folgt kein Automatismus in die eine oder andere Richtung – aber eine rein formale Freigabeschaltfläche erfüllt diese Anforderung jedenfalls nicht von selbst. Maßgeblich sind Risikoadäquanz, tatsächliche Eingriffsmöglichkeit und der konkrete Anwendungskontext. Genau darauf zahlen Risikoklassen, begrenzte Fähigkeiten, unabhängige Autorisierung und eine vertrauenswürdige Freigabemaske ein.

Für **NIS2**-pflichtige Unternehmen ist die Verbindung vor allem über Governance und Cyber-Risikomanagement relevant: Die Leitungsorgane müssen die Risikomanagementmaßnahmen billigen und ihre Umsetzung überwachen (Art. 20). Die Schlusskette daraus ist kurz – unbekannte autonome Agenten sind unbekanntes Cyber-Risiko, und was nicht inventarisiert ist, lässt sich nicht steuern. Damit wird ein Agent Inventory unmittelbar governance-relevant.

DORA verlangt von Finanzunternehmen, IKT-Drittparteienrisiken als integralen Bestandteil des IKT-Risikomanagements zu behandeln und die zugehörigen Vertragsbeziehungen zu dokumentieren und zu steuern. Für Agentic AI ist der Anschluss direkt: Wird agentische Funktionalität durch einen SaaS-Anbieter in einen kritischen Geschäftsprozess eingebracht, verändert sie die bestehende IKT-Drittparteienabhängigkeit und gehört in deren Risikobewertung – auch dann, wenn die Agentenfunktion nicht gesondert beschafft wurde.

> **Einordnung:** Dieses Papier ordnet ein und ersetzt keine Rechtsberatung. Anwendungsbereich, Fristen und Umsetzungsstand sind zum Zeitpunkt der Anwendung zu prüfen; die Geltung einzelner Teile des AI Acts ist zeitlich gestaffelt.

Es ist keine Aufgabe eines einzelnen Teams

Agentic AI berührt IAM, PAM, Application Security, Cloud Security, Data Security, SOC, Detection Engineering, Third-Party Risk, GRC und Enterprise Architecture. Die Herausforderung ist ihre Verbindung.

Das AI-Team verantwortet Modellqualität und Evaluation. Der Geschäftsbereich besitzt Prozess und Nutzen. Security definiert die Sicherheitsarchitektur und kontrolliert die technischen Grenzen. Die Geschäftsleitung akzeptiert das verbleibende Risiko. Das Entscheidende ist, diese Zuständigkeiten explizit zu machen, bevor der erste Vorfall sie implizit klärt.

Vieles davon existiert bereits

IAM, OAuth, Workload Identity, PAM, API Gateway, Policy Engine, Secrets Management, DLP, SIEM, SOAR, Sandboxing und Cloud IAM sind in den meisten Organisationen vorhanden. Die Aufgabe besteht darin, diese Kontrollen an der richtigen Stelle zusammenzuführen.

ABB. 10 **Die Kontrollkette.** Jede Stufe ist ein eigenes Arbeitsgebiet – und ein eigenes Paper der Serie.

Der praktische Blocker ist deshalb selten die Technologie. Er ist die Frage, **wer die Agenten-Policies und -Budgets schreibt, testet und pflegt** – und in welchem Team diese Zuständigkeit liegt. Wer das nicht früh klärt, baut einen Policy Enforcement Point, den niemand aktuell hält.

9

Sechs Fragen, die jetzt zu beantworten sind

Die einzige Checkliste in diesem Papier – für die kritischsten Agenten zuerst.

Erstens: Welche Agenten führen bereits Aktionen aus?
Ein Agent Inventory ist kein Verwaltungsprojekt, sondern Voraussetzung für jede weitere Kontrolle. Besonders relevant sind Systeme mit Schreibrechten, mit Zugriff über mehrere Sicherheitsdomänen hinweg oder mit externen Kommunikationswegen – einschließlich der Agenten, die per SaaS-Update in den eigenen Tenant geliefert wurden und die niemand beschafft hat.

Zweitens: Unter welcher Identität handeln sie?
Für jeden produktiven Agenten sollte nachvollziehbar sein, welcher technische Principal handelt und welche menschliche oder geschäftliche Autorität dahintersteht. Wenn die Antwort „unter dem Token des Benutzers“ lautet, ist das die erste Baustelle.

Drittens: Welche Fähigkeiten besitzen sie?
Tool-Inventare sind für Agenten so wichtig wie Berechtigungsinventare für Benutzer. Nicht benötigte Fähigkeiten gehören entfernt, nicht per Prompt untersagt. Und: Gibt es einen erreichbaren Pfad durch Agenten und Werkzeuge, der die drei Eigenschaften aus Kapitel 4 vereint?

Viertens: Welche Aktionen benötigen Autorisierung zur Laufzeit – und ist diese Grenze umgehbar?
Kategorien für kritische, schwer umkehrbare oder extern wirksame Aktionen definieren und dafür unabhängige Enforcement Points etablieren. Und prüfen, ob daneben noch eine Shell, ein roher Netzwerkzugang oder ein direktes Credential existiert.

Fünftens: Wie groß darf ein Agent insgesamt werden?
Nicht technisch, sondern autoritativ. Welchen maximalen kumulativen Schadensradius darf ein einzelner Agent, ein Workflow oder ein delegierter Auftrag besitzen? Die Frage lautet nicht „Darf er eine Bestellung anlegen?“, sondern „Könnte ein kompromittierter Agent hunderttausend Bestellungen anlegen?“ Daraus folgen Aktions-, Werte-, Daten- und Zeitbudgets.

Sechstens: Wie stoppen wir einen Agenten?
Credential Revocation, Tool Disablement, Netzwerkisolation und Rollback müssen vor dem Vorfall entworfen und getestet werden, nicht während seiner Bearbeitung.

Ein Unternehmen, das diese sechs Fragen für seine kritischsten Agenten beantworten kann, steht besser da als eines mit umfangreichen AI-Guidelines und unbekannten produktiven Agenten.

ABSCHLUSS

Agentic AI macht aus einem bekannten Problem eine neue Systemklasse. Das Modell bleibt wichtig – seine Sicherheit, seine Kontextdaten, seine Robustheit gehören weiterhin zu einer belastbaren Architektur.

Die entscheidende Veränderung beginnt dort, wo das System handeln kann. Von diesem Moment an reicht die Frage „Wie sorgen wir dafür, dass das Modell das Richtige tut?“ nicht mehr aus.

Ein Agent darf weder wie ein Benutzer noch wie ein klassischer Dienst behandelt werden. Er ist ein Workload mit dynamisch gewähltem Handlungsverlauf und delegierter Autorität. **Deshalb muss seine Identität explizit, seine Autorität begrenzt, jede folgenreiche Aktion unabhängig vermittelt und seine kumulative Wirkung budgetiert sein.**

Darin liegt der Übergang von Model Security zu Action Security.

Wir müssen nicht jede Fehlentscheidung eines Agenten verhindern können. Wir müssen kontrollieren, was aus ihr werden darf.

WEITERFÜHRENDE QUELLEN

OWASP GenAI Security Project – *Top 10 for Agentic Applications*. Beschreibt zentrale Risikokategorien für agentische Anwendungen, darunter Goal Hijacking, Tool Misuse, Identity and Privilege Abuse, Memory and Context Poisoning, unsichere Agent-to-Agent-Kommunikation, Cascading Failures und Rogue Agents.

OWASP GenAI Security Project – *Agentic Security Initiative*. Ergänzende Guidance zu MCP, Threat Modeling und Governance.

MITRE ATLAS – *Adversarial Threat Landscape for Artificial-Intelligence Systems*. Wissensbasis zu Angriffstaktiken gegen AI-Systeme, mit eigener Plattformklassifikation für Agentic AI und Techniken rund um Tool Invocation, Context Poisoning und Tool Poisoning.

NIST – *AI Risk Management Framework (AI RMF 1.0)*, NIST AI 100-1, 2023, sowie *Generative AI Profile*, NIST AI 600-1, 2024.

NIST – *SP 800-207 Zero Trust Architecture*, 2020, sowie *SP 800-207A* zu granularen Policies auf Basis von Anwendungs- und Dienstidentitäten.

IETF – *RFC 8693, OAuth 2.0 Token Exchange*, 2020. Delegationsdarstellung über `act` und `may_act`.

IETF – *RFC 9396, OAuth 2.0 Rich Authorization Requests*, 2023. Feingranulare Autorisierungsinformationen über `authorization_details`.

IETF – *RFC 9449, OAuth 2.0 Demonstrating Proof of Possession (DPoP)*, 2023. Senderbindung von Tokens.

Simon Willison – *The lethal trifecta for AI agents*, 2025.

Google DeepMind – *CaMeL: Defeating Prompt Injections by Design*, 2025.

Saltzer / Schroeder – *The Protection of Information in Computer Systems*, 1975. Ursprung des Prinzips der vollständigen Vermittlung.

Abbildungsverzeichnis

Die Serie

01	Von Model Security zu Action Security	DIESES PAPIER
02	Agent Inventory: Was läuft eigentlich bei uns?	IN VORBEREITUNG
03	Identität und Delegation für KI-Agenten	IN VORBEREITUNG
04	Least Capability	IN VORBEREITUNG
05	Runtime Authorization für autonome Systeme	IN VORBEREITUNG
06	MCP und Tool Security	IN VORBEREITUNG
07	Prompt Injection ist ein Aktionsproblem	IN VORBEREITUNG
08	Der Agent Action Ledger	IN VORBEREITUNG

Alle Papers erscheinen unter cycademy.de/agentic-ai-security-papers. Verwandt, außerhalb der Serie: *NIS2 trifft Agentic AI* (August 2026).

NÄCHSTER SCHRITT

Grenzen bauen, nicht formulieren.

Ein Prompt beschreibt, was ein Agent tun soll. Eine Grenze entscheidet, was er tun kann. Im zweitägigen Training **„AI Agents für Security Management“** bauen Sie handelnde Agenten selbst – vollständig lokal, ohne eine Zeile Code – und ziehen die Grenzen aus diesem Papier direkt ein: eigene Identität, begrenzte Delegation, minimale Fähigkeiten, Freigabe an den Stellen, an denen sie wirklich trägt. Sie gehen mit lauffähigen Werkzeugen und einem 90-Tage-Plan nach Hause.

MEHR ERFAHREN UND PLATZ SICHERN

cycademy.de/ai-agents-security-management

Cycademy · Train.Coach.Guide. – Agentic AI Security Papers, Nr. 01 „Von Model Security zu Agentic Action Security“, Version 1.0, Stand August 2026.

Quellen (Auswahl): OWASP GenAI Security Project, Top 10 for Agentic Applications · MITRE ATLAS · NIST AI RMF 1.0 (2023) · NIST SP 800-207/-207A · IETF RFC 8693, RFC 9396, RFC 9449 · Simon Willison (2025) · Google DeepMind, CaMeL (2025) · Saltzer/Schroeder (1975). Framework-Stände bitte zum Zeitpunkt der Anwendung prüfen.