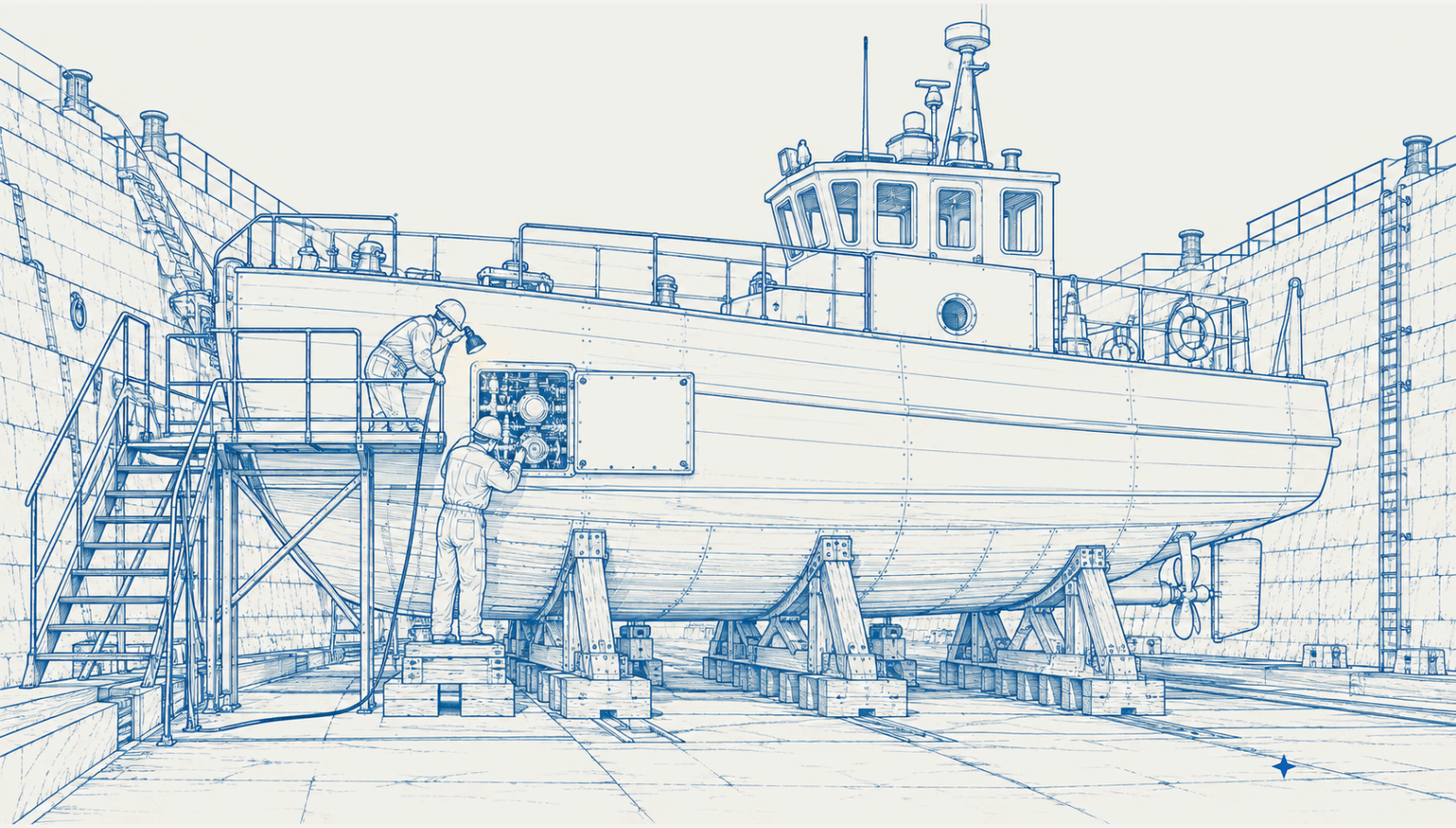
GOVERNED INTELLIGENCE · WHITEPAPER

Sicherheitsprüfungen mit KI-Agenten

Wie aus einem Verdacht ein belastbarer Befund wird

Timotheos Samartzidis
30. Oktober 2026 · Fassung 3.10



Vom Verdacht auf Fremdzugriff zum Prüfauftrag

Ein Kunde öffnet ein Dokument in einem Webportal. Die Anwendung prüft seine Anmeldung, nimmt die Dokumentnummer entgegen und liefert die Datei aus. Ein KI-Agent untersucht den zugehörigen Quellcode auf Sicherheitsfehler und meldet: Ein angemeldeter Kunde könnte durch eine andere Dokumentnummer auf die Unterlagen eines fremden Kunden zugreifen.

Das wäre eine erhebliche Schwachstelle. Zunächst ist es eine Behauptung mit mehreren Voraussetzungen. Kann der Kunde die Nummer frei wählen? Erreicht seine Anfrage die fragliche Funktion? Prüft ein vorgelagerter Dienst bereits, wem das Dokument gehört? Und verhindert die Datenbank den Zugriff möglicherweise an einer Stelle, die im ersten Codeausschnitt fehlt?

An diesem fiktiven Beispiel entwickelt das Whitepaper einen Prüfprozess für die Anwendungssicherheit. Wenn Sie Entwicklung, Architektur oder Sicherheit verantworten, soll er Ihnen helfen, den Beitrag von Agenten einzuordnen und einen ersten Einsatz vorzubereiten. Die Untersuchung muss vom Verdacht über belastbare Belege zu einer verantworteten Behandlung führen. Dazu gehören wiederholbare Analysen, eine gesonderte Gegenprüfung und, wo erforderlich, kontrollierte Tests.

Schon im Portal sind zwei Fragen auseinanderzuhalten. Die **Authentisierung** klärt, wer ein Benutzer ist. Die **Autorisierung** entscheidet, was dieser Benutzer mit einem bestimmten Objekt tun darf. Die Anmeldung kann korrekt funktionieren, während die Prüfung des einzelnen Dokuments fehlt. Solche Fehler liegen im Zusammenspiel der Anwendung und lassen sich aus einem einzelnen Codeausschnitt oft nicht beurteilen.

Der folgende Vorschlag verbindet deshalb Werkzeuge, Kontext und menschliche Verantwortung. Er behandelt Agent Skills für Coding-Agenten, keine Pentesting-Zertifizierungen. Fachliche und technische Vertiefungen stehen jeweils bei der Frage, die sie erklären. Beobachtungen aus einem eigenen Versuch mit künstlichem Testcode begleiten die jeweiligen Untersuchungsschritte. Sie zeigen, welche Ergebnisdateien entstehen und wo die Ausführung an Grenzen stößt. Die Auswahl und der Einführungsplan sind eigene Empfehlungen; ihre Wirksamkeit im Zielsystem muss ein Pilot, also ein begrenzter Einführungsversuch, erst zeigen. Betriebsüberwachung und die Reaktion auf reale Sicherheitsvorfälle bleiben mit den dafür verantwortlichen Teams zu verbinden.

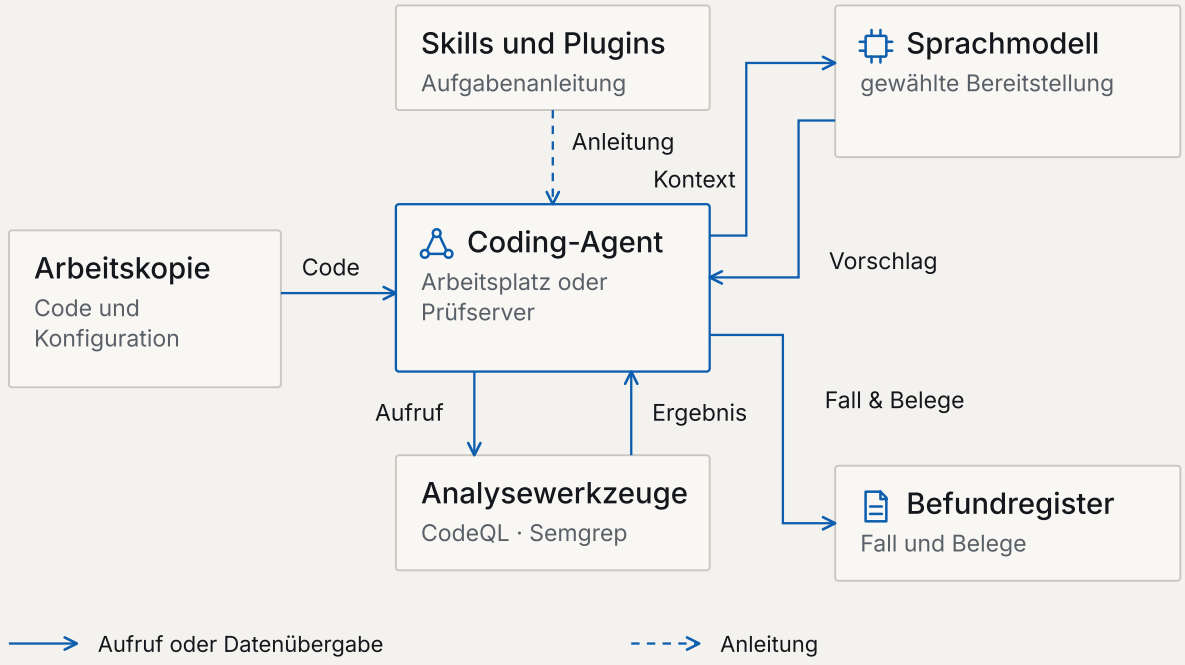
Wie ein Coding-Agent arbeitet und welche Werkzeuge ihn unterstützen

Ein Coding-Agent verbindet ein Sprachmodell mit Zugriffen auf Dateien und Werkzeuge. Das Modell interpretiert den Auftrag und schlägt nächste Schritte vor. Die Agentensoftware führt zulässige Werkzeugaufrufe aus, verarbeitet deren Ergebnisse und setzt die Bearbeitung fort. So kann sie eine Funktion lesen, deren Aufrufer suchen und anschließend eine Analyse starten. Codex, Claude Code und Cursor sind mögliche Arbeitsumgebungen dafür.

Die **Arbeitskopie** ist das für die Untersuchung bereitgestellte Verzeichnis mit Quellcode und Konfiguration. Das **Befundregister** ist eine zugriffsbeschränkte gemeinsame Fallliste, etwa im Ticketsystem. Es verbindet Behauptung, Belege, Status, Verantwortliche und Behandlung.

Ein **Skill** liefert eine Aufgabenanleitung, gegebenenfalls mit Skripten und weiteren Ressourcen. Ein **Plugin** bündelt Erweiterungen für die jeweilige Agentensoftware. Trail of Bits Security Skills ist eine veröffentlichte Sammlung solcher Sicherheitsabläufe. Abbildung 1 zeigt, wie diese Anleitungen, das Modell und die ausführbaren Werkzeuge zusammenwirken. Trail of Bits

1 · ZUSAMMENSPIEL DER BAUSTEINE



Logische Interaktion; konkrete Dienste und Datenwege werden im Projekt festgelegt.

Abbildung 1: Der Agent vermittelt zwischen Modellvorschlägen und Werkzeugaufrufen. Skills geben den Ablauf vor; die Umgebung begrenzt die Rechte.

Das Sprachmodell verarbeitet den freigegebenen Kontext beim gewählten Modelldienst oder in einer selbst betriebenen Bereitstellung. CodeQL und Semgrep sind eigenständige Analyseprogramme in der Prüfumgebung. Sie benötigen Quelldateien und Konfiguration und speichern Analyseergebnisse.

Der Agent verwendet somit die Anleitung eines Skills und die Ergebnisse ausführbarer Werkzeuge. Eine Skillinstallation richtet diese Programme nicht automatisch ein; ihre Einrichtung wird separat geprüft.

FACHLICHE VERTIEFUNG

Werkzeuge für den Einstieg priorisieren

Die folgende Priorisierung ordnet die Auswahl für den Einstieg. Sie ist eine fachliche Empfehlung, kein standardisierter Leistungsvergleich. Projektbeschreibungen belegen den dokumentierten Funktionsumfang, nicht die gemessene Erkennungsleistung im eigenen System.

Rang	Auswahl	Beitrag
1	Trail of Bits Security Skills	Vertiefte Code- und Datenflussanalyse, Gegenprüfung, Varianten und ergänzende Fuzzing-Abläufe.
2	Strix	Autorisierte aktive Untersuchung mit konkreten Exploit-Nachweisen.
3	CodeQL / Semgrep	Systematische, wiederholbare Quellcodeprüfung.
4	OpenAI `security-threat-model`	Schutzwerte, Angriffswege und Vertrauensgrenzen am Code entwickeln und mit Betriebswissen ergänzen.
5	Trail of Bits `fp-check`	Vorhandene Verdachtsfälle mit gezielten Widerlegungsversuchen prüfen.
6	Shannon	Quellcodeanalyse mit aktiver Validierung verbinden; Betriebsarten unterscheiden.
7	NVIDIA SkillSpector	Skills auf verdächtige Anweisungen und mögliche Datenabflüsse untersuchen.

Für den ersten Einsatz empfehle ich den statischen Kern aus `static-analysis` und `fp-check`, ergänzt um Kontextaufbau, Varianten- und Änderungsprüfung. Strix folgt für gezielt erlaubte aktive Tests. Bevor diese Auswahl arbeiten kann, braucht das Team einen bestimmten Gegenstand und einen begrenzten Auftrag.

Bevor der Agent sucht, braucht er einen Auftrag

Ein **Repository** ist die gemeinsame Ablage des Quellcodes und seiner Änderungsgeschichte. Eine Revision, meist durch einen Git-Commit bezeichnet, hält den untersuchten Codestand fest. Die relevante Betriebskonfiguration wird zusätzlich dokumentiert. Ohne diese Festlegung kann die Entwicklung den Fehler bereits verändert haben, während der Agent noch den alten Stand erklärt.

Beim Portal interessiert außerdem, welche Dienste Anfragen annehmen, wo Kundenidentitäten entstehen und wer die Dokumentberechtigung durchsetzt. Eine Programmierschnittstelle, kurz **API**, ist ein definierter Zugang, über den ein anderes Programm eine Funktion aufruft. Ihre Erreichbarkeit verändert die möglichen Angriffswege.

Ein **Bedrohungsmodell** ordnet diese Informationen: Was soll geschützt werden, wer könnte es angreifen, über welche Zugänge und mit welchen Fähigkeiten? Im Beispiel sind die vertraulichen Kundendokumente der Schutzwert. Eine **Vertrauensgrenze** liegt dort, wo Daten oder Handlungen aus einem anders berechtigten Bereich übernommen werden. Zwischen Kunde und Portal muss die Anwendung selbst durchsetzen, welche Dokumente zum Kunden gehören. Eine gelieferte Dokumentnummer beweist diese Zugehörigkeit nicht.

Der OpenAI-Skill `security-threat-model` unterstützt die Bestandsaufnahme am Quellcode. Das Anwendungsteam bestätigt den Betriebskontext, weil das Repository tatsächliche Netzfreigaben, Identitätsdienste und Schutzmaßnahmen nicht zuverlässig vollständig zeigt. Auch Systeme, die Code bauen und testen, sind eigene Angriffsflächen mit eigenen Rechten. Threat-Modeling-Skill

PRAXISANLEITUNG · AGENTENAUFTRAG

Ein Bedrohungsmodell erarbeiten

Beginnen Sie mit einem ausdrücklichen Auftrag an den geladenen Skill: „Erstelle für diese Revision ein Bedrohungsmodell. Frage fehlenden Betriebskontext nach und trenne Codebelege, bestätigte Annahmen und Hypothesen.“ Übergeben Sie Architektur, Datenklassen und bekannte Zugänge. Das Ergebnis soll Komponenten, Datenflüsse, Schutzwerte, Bedrohungen, Maßnahmen und offene Fragen verknüpfen. Erst die bestätigten Angaben tragen den nächsten Prüfauftrag.

Ihr Team benennt vor dem Lauf die Verantwortlichen. Der Anwendungseigentümer bestätigt Zweck, Betrieb und Prüfbereich. Die fachliche Sicherheitsverantwortung, häufig **AppSec** für Application Security genannt, bewertet Fälle und pflegt das Befundregister. Die Plattformverantwortlichen richten Zugriffe und Ausführungsgrenzen ein. In kleinen Teams können Personen mehrere Aufgaben übernehmen; ihre Entscheidungen bleiben nachvollziehbar.

FACHLICHE VERTIEFUNG

Was am Portalbeispiel tatsächlich geprüft wurde

Um den Prüfweg an konkreten Ergebnissen zu erklären, wurde ein eigener kleiner Testaufbau untersucht. Er enthält zwei künstliche Kunden mit je einem Dokument. In zwei Abruffunktionen wurde die Eigentümerprüfung absichtlich entfernt; eine dritte behält sie. Für einen Dokumentassistenten ergänzt eine Werkzeugbeschreibung mit der zugehörigen ausführenden Funktion, dem Handler, den Fall. Die Ausgangsrevision, die fehlerhafte Änderung und eine korrigierte Kopie sind getrennt festgehalten.

Für die bedingte Risikobewertung bestätigte der Auftraggeber zusätzlich die Betriebsannahme eines über das Internet erreichbaren Portals mit vertraulichen Kundendokumenten. Ein solches Portal wurde weder aufgebaut noch angegriffen. Die tatsächlichen Funktionsaufrufe verwendeten ausschließlich künstliche Daten; ein vollständiger Server für den Dokumentassistenten war ebenfalls nicht Teil des Aufbaus. Wenn im Folgenden vom Probelauf die Rede ist, gilt dieser begrenzte Umfang. Der fiktive Betriebsfall veranschaulicht weitergehende Entscheidungen, die der lokale Versuch nicht nachweisen kann.

Die Fehler und erwarteten Antworten waren bekannt. Geprüft wurden die 18 im Whitepaper konkret genannten Abläufe und Werkzeuge, nicht alle Inhalte ihrer verlinkten Sammlungen. Der Versuch zeigt beobachtete Ergebnisse, Anpassungen und fehlende Voraussetzungen. Er misst keine Erkennungsleistung an unbekannten Schwachstellen.

Nicht alle 18 Einträge sind vollständig getestet. Nach den ergänzenden Läufen wurden zwölf im jeweils bezeichneten Umfang ausgeführt, drei teilweise; zwei blieben blockiert und ein Lauf scheiterte technisch. Zu den ausgeführten Schritten gehören auch Agentenmethoden und angepasste Abläufe. Eine native Installation und vollständige Ausführung aller Codex- oder Claude-Plugins ist damit nicht nachgewiesen.

Die Statusangaben beschreiben die Ausführung: „Ausgeführt“ heißt, dass der benannte Prüfschritt mit Artefakt stattfand; „teilweise“ weist eine fehlende Teilstufe aus, „blockiert“ eine nicht erfüllte Voraussetzung und „gescheitert“ einen begonnenen, technisch abgebrochenen Lauf. Keiner dieser Werte ist für sich ein Urteil über die Sicherheit der Anwendung.

Ein **Ergebnisartefakt** ist eine dauerhaft gespeicherte Datei, an der sich ein Untersuchungsschritt nachvollziehen lässt. Dazu gehören Agentenberichte ebenso wie native, vom Werkzeug selbst geschriebene Scannerdateien und Laufprotokolle. Ihre Aufgaben unterscheiden sich: Ein Befundbericht erklärt einen Verdacht; ein Laufprotokoll zeigt, ob die Untersuchung vollständig ausgeführt wurde. Die folgenden Kapitel erläutern diese Dateien bei dem Verfahren, das sie hervorbringt. Relative Dateipfade beziehen sich auf die verlinkte Testablage. Ihr Testbericht enthält die vollständige Zuordnung mit Quellenständen, Befehlen und Originalausgaben; Namen eigener Berichte sind Konventionen dieses Versuchs.

Das erste solche Ergebnis ist `runs/threat-model/fixture-threat-model.md`. Der mit `security-threat-model` erstellte Bericht verbindet Komponenten, Datenflussdiagramm, Schutzwerte und Annahmen mit drei Bedrohungen, ihrer bedingten Priorität und Maßnahmen. Er hält auch fest, welche Bewertung nur unter der bestätigten Annahme des Internetbetriebs gilt. Damit ist sichtbar, auf welcher Grundlage der anschließende Prüfauftrag entsteht.

Für Ihre eigene Durchführung stehen die Handlungsanweisungen in blau markierten Kästen mit der Bezeichnung „Praxisanleitung“, direkt beim jeweiligen Verfahren. Der Zusatz am Kasten unterscheidet Agentenauftrag, Werkzeugaufruf und Einrichtung. Die tatsächlich beobachteten Dateien, Resultate und Grenzen werden beim jeweiligen Verfahren ausdrücklich beschrieben. Ein erwartetes Artefakt im Kasten ist noch kein erzeugtes Testergebnis. Die Praxisanleitung zu allen 18 Einträgen hält ergänzend Voraussetzungen, Aufträge, Befehle, Erfolgskriterien und verbleibende Grenzen zusammen. Die Shell-Beispiele setzen nach der jeweiligen Werkzeuginstallation Bash/WSL und das Hauptverzeichnis einer entbehrlichen Kopie der Testablage voraus, sofern der Abschnitt keine andere Umgebung nennt. SkillSpector verwendet ausdrücklich den getesteten Windows-Python-Einstieg. `eigener-lauf/` ist ein neuer Ausgabeordner, damit die aufgezeichneten Originale erhalten bleiben.

FACHLICHE VERTIEFUNG

Den ersten Lauf konkret beauftragen

PRAXISANLEITUNG · ARBEITSAUFTRAG

Den ersten Prüfauftrag begrenzen

Für den ersten Portal-Audit kann der Auftrag lauten: „Untersuche die freigegebene Revision darauf, ob ein angemeldeter Kunde fremde Dokumente lesen kann. Verändere den Anwendungsquellcode nicht. Erfasse erreichbare Zugänge, den Datenfluss und wirksame Berechtigungsprüfungen. Dokumentiere Verdachtsfälle, offene Annahmen und nicht untersuchte Bereiche. Erzeuge Analyseartefakte nur in der vereinbarten Ablage. Starte keine Tests gegen laufende Dienste und keine Programme außerhalb der freigegebenen Werkzeugliste.“

Das Team ergänzt Repository und Commit, betrachtete Funktionen, bestätigte Betriebsannahmen, erlaubte Werkzeuge, Datenwege, Ablage und Aufwandgrenze. Ein Lauf liefert die vereinbarten Ergebnisse oder einen benannten Abbruchgrund. Fehlt etwa der Code des vorgeschalteten Berechtigungsdienstes, bleibt dessen Wirkung ungeklärt. Der Agent darf sie weder als wirksam voraussetzen noch ohne Belege als fehlend melden.

Dieser Auftrag untersucht zunächst den Dokumentabruf. Weitere Prüfbereiche werden ausdrücklich beauftragt und mit passenden Verfahren bearbeitet. Berichte und Analysedaten dürfen außerhalb der geschützten Anwendungsarbeitskopie entstehen. Builds, also die Aufbereitung des Quellcodes und seiner Abhängigkeiten für die Ausführung, benötigen eine gesondert begrenzte Ausführungserlaubnis. „Nur lesen“ beschreibt eine technisch einzurichtende Berechtigung. An dieser Grenze beginnt die eigentliche Analyse.

Quellcode, Datenfluss und Scannerergebnisse verbinden

Das prüfende Team verfolgt die Dokumentnummer von der Anfrage bis zur Dateiausgabe und sucht die wirksame Berechtigungsprüfung. Dieser Weg durch Variablen, Funktionen und Dienste heißt **Datenfluss**. Die Folge der Funktionsaufrufe bildet die Aufrufkette. Beide Sichten werden benötigt: Eine geprüfte Variable hilft wenig, wenn die Anwendung später einen anderen Wert verwendet.

Scanner übernehmen automatisierte Prüfungen des Quellcodes. Diese **statische Sicherheitsanalyse**, kurz SAST, erfordert keinen Angriff auf die laufende Zielanwendung. CodeQL bereitet Code in einer Analysedatenbank auf und untersucht ihn mit Abfragen. Semgrep sucht mit Regeln nach verdächtigen Mustern und kann abhängig von Regel, Sprache und Edition ebenfalls Datenflüsse untersuchen. Eine **Taint-Analyse** verfolgt, ob eine beeinflussbare Eingabe bis zu einer sicherheitsrelevanten Verwendung gelangt und welche Prüfungen dazwischenliegen. Regeln und Datenflussmodelle bestimmen wesentlich, was sichtbar wird. CodeQL, Semgrep

Eine Regel könnte die frei übernommene Dokumentnummer auffällig finden. Ob eine andere Stelle den Zugriff absichert, bleibt eine Kontextfrage. Der Agent liest zusammengehörige Funktionen und ihre Aufrufer und entwickelt prüfbare Hypothesen. So kann er Zusammenhänge untersuchen, die vorhandene Regeln nicht ausdrücken. Er kann ebenso Schutzfunktionen missverstehen oder Wege übersehen. Die Scannerbasis und die agentische Untersuchung ergänzen sich; beide brauchen eine sichtbare Abdeckung.

TECHNISCHE VERTIEFUNG

Die sieben Trail-of-Bits-Plugins

Eine `SKILL.md` beschreibt die Aufgabenanleitung; einzelne Plugins verwenden andere Einstiegspunkte. So besteht `insecure-defaults` im untersuchten Stand aus einer Befehlsdatei und einem Workflow. Für den Einstieg verwende ich die Bewertungen „Pflicht“, „sehr hoch“ und „hoch“. Die Einordnung gilt als Priorisierung für allgemeine Enterprise-Codebestände; der konkrete Einsatz richtet sich nach der Aufgabe.

Rang / Priorität	Plugin	Funktion und Grenze
1 · Pflicht	`static-analysis`	Enthält `codeql`, `semgrep` und `sarif-parsing`. SARIF ist ein standardisiertes Austauschformat für Ergebnisse statischer Analysen. Der Ablauf führt Analysen aus und verarbeitet Ergebnisse; Programme, Regeln und Datenbanken sind separat einzurichten.
2 · Pflicht	`audit-context-building`	Dokumentiert Funktionen, Annahmen, Abhängigkeiten und Datenflüsse. Liefert Kontextdossiers und offene Fragen; dieser Schritt sucht oder behebt noch keine Fehler und vergibt keine Schwere.

Rang / Priorität	Plugin	Funktion und Grenze
3 · Pflicht	`fp-check`	Prüft konkrete vorhandene Kandidaten mit Aufrufern, Schutzbedingungen und Gegenargumenten. Das begründete Urteil ersetzt keine fehlenden Belege.
4 · sehr hoch	`variant-analysis`	Leitet aus einer bekannten Ursache eine enge Suchsignatur ab, erweitert sie schrittweise und bewertet neue Kandidaten. Kein initiales Suchen ohne Ausgangsfall.
5 · sehr hoch	`differential-review`	Prüft Git-Änderungen mit Historie, betroffenen Aufrufern, Testabdeckung und Reichweite. Die sichtbare Änderung allein kann unzureichend sein.
6 · hoch	`insecure-defaults`	Untersucht Ersatzgeheimnisse, Standardzugangsdaten, bei Fehlern freigebende Prüfungen (Fail-open), schwache Kryptographie, zu weite Rechte und Debug-Datenlecks. Kandidaten werden verifiziert.
7 · hoch	`semgrep-rule-creator`	Entwickelt Regeln testgetrieben mit unsicheren Positivfällen, sicheren Negativfällen, Prüfung der Syntaxstruktur, Iteration bis alle Tests bestehen und Abschlusskontrolle.

Quellen: statische Analyse, Kontextaufbau, Varianten, Regelentwicklung, unsichere Voreinstellungen.

Ein erster Gesamtaudit folgt im Vorschlag der Reihenfolge `audit-context-building → semgrep → codeql → fp-check → variant-analysis`. Die Kontextaufnahme kommt zuerst; Varianten folgen aus verstandenen Ausgangsfällen. Diese Reihenfolge ist eine praktische Wahl, keine Vorgabe für jedes Repository.

PRAXISANLEITUNG · AGENTENAUFTRAG

Den Codekontext aufbauen

Mit der festgelegten Revision kann der Auftrag an `audit-context-building` lauten: „Baue für diese Funktionen zunächst ausschließlich Kontext auf. Beschreibe Eingaben, Aufrufer, Zustandsänderungen, Abhängigkeiten und Annahmen; vergib noch keine Sicherheitsbefunde.“ Verlangen Sie ein Dossier mit einem Eintrag je untersuchter Funktion und einer Liste offener Fragen. Verwenden Sie dafür die im jeweiligen Host unterstützte Delegation.

Wie dieser Kontext greifbar wird, zeigt der Probelauf: `audit-context-building` hinterließ im Ordner `runs/context/audit-context/` ein `DOSSIER.md`, sechs einzelne Funktionsbeschreibungen unter `functions/` und `execution.json`. Dafür wurden sechs getrennte Codex-Funktionsagenten eingesetzt; die native Claude-Workflow-API wurde durch diesen Ablauf ersetzt. Das Dossier hält Annahmen und Datenflüsse fest, ohne bereits ein Sicherheitsurteil zu vergeben. Die verwendeten Trail-of-Bits-Workflows sind auf konkrete Revisionen festgehalten.

TECHNISCHE VERTIEFUNG

Scanner konfigurieren und Ergebnisse lesen

Bei CodeQL werden Sprache, Datenbankextraktion, Build-Modus und Abfragesammlung festgehalten. Für kompilierte Sprachen kann der Build bestimmen, welche Dateien in die Datenbank gelangen. Projektinterne Eingaben, kritische Operationen und Schutzfunktionen können zusätzliche Modelle erfordern. Extraktions- und Build-Diagnosen sowie nicht erfasste Bereiche gehören zum Laufbericht. Ein erfolgreicher Datenbankaufbau oder null Meldungen belegt keine vollständige Untersuchung. Build-Modi

Bei Semgrep werden Engine, Edition, Sprache und Regelpakete dokumentiert. Community- und Pro-Funktionen unterscheiden sich besonders bei Analysen über Funktions- und Dateigrenzen; die Unterstützung hängt vom Regeltyp und der Sprache ab. Fehlgeschlagene, übersprungene, teilweise oder für den relevanten Code wirkungslose Läufe werden ausgewiesen. `--metrics=off` ersetzt keine Prüfung anderer Netz- und Plattformfunktionen. Produktgrenzen, Semgrep-Skill

PRAXISANLEITUNG · WERKZEUGAUFRUF

Semgrep-Basislauf starten und Ergebnisse prüfen

Nach der Einrichtung lässt sich ein einzelner Semgrep-Lauf so beginnen. JSON speichert die Ergebnisse als strukturierte Textdaten; SARIF dient dem standardisierten Austausch. Wählen Sie zusätzliche Regeln ausdrücklich nach Sprache und Prüfauftrag. Der kurze Aufruf unten ersetzt nicht die sechs Regelläufe des dokumentierten Versuchs.

```
mkdir -p eigener-lauf/semgrep
semgrep scan --config p/python --metrics=off \
  --json-output eigener-lauf/semgrep/results.json \
  --sarif-output eigener-lauf/semgrep/results.sarif fixture/
```

Lesen Sie danach Laufstatus, Fehlermeldungen und untersuchte Dateien zusammen mit den Findings. Eine geschriebene SARIF-Datei beweist noch keinen vollständigen Lauf.

Erwartetes Artefakt: JSON und SARIF im neuen Ausgabeordner; Laufstatus, Fehler und untersuchte Dateien werden mitgelesen.

Im Probelauf wurde diese Unterscheidung sichtbar. Semgrep erzeugte im Ordner `runs/semgrep/` sechs native JSON-/SARIF-Paare unter `raw/`; `rulesets.json` beschreibt die Regelwahl, `scans.json` den Laufstatus und `results/results.sarif` die zusammengeführten Ergebnisse. Die Basisregeln meldeten keine Treffer. Zwei der sechs Regelläufe waren jedoch wegen Regelproblemen unvollständig, Pro war nicht verfügbar. Die Dateien belegen damit durchgeführte Analysen und ihre Grenzen. Die bekannte Berechtigungslücke blieb in dieser Basisprüfung ungemeldet.

PRAXISANLEITUNG · AGENTENAUFTRAG

CodeQL-Analyse mit dokumentierter Abdeckung beauftragen

Für CodeQL beginnt der Ablauf mit dem Kommandozeilenprogramm, der CLI, und einer passenden Sprachumgebung. Der Agentenauftrag lautet: „Baue für diese Revision eine neue Datenbank, prüfe ihre Extraktionsqualität, untersuche nötige projektspezifische Datenmodelle und analysiere mit einer ausdrücklich ausgewählten Abfragesuite. Dokumentiere ausgelassene Bereiche.“ Eine Suite ist eine festgelegte Abfragesammlung. Prüfen Sie vor der Analyse, welche Abfragen die Suite tatsächlich auflöst; eine leere Datenbank oder eine ungewollt gefilterte Sammlung kann ebenfalls null Meldungen liefern.

Erwartetes Artefakt: Analysedatenbank, Extraktions- und Qualitätsdiagnose, festgelegte Suite, SARIF und dokumentierte Abdeckung.

Im Erstlauf fehlte CodeQL. Anschließend wurde CodeQL 2.27.3 projektlokal eingerichtet und an einem getrennten öffentlichen MIT-Regressionsbeispiel demonstriert. Unter `runs/followup/codeql/` entstanden `codeql.db/`, `build.log`, Qualitätsdiagnose, die explizite Suite `raw/demo.qls` und `raw/results.sarif`. Die Datenbankprüfung erfasste eine Projektdatei mit 44 Baseline-Codezeilen und null Extraktionsfehlern. Die einzelne Abfrage meldete 13 im Beispiel markierte Stellen; eine davon ist im Upstream ausdrücklich als `SPURIOUS`, also als Fehlalarm, gekennzeichnet.

Das ist eine native Werkzeugdemonstration, kein vollständiger CodeQL-Skill-Test. Die breite Suite- und Modellierungsfolge blieb ungetestet; der Portalcode wurde nicht mit CodeQL untersucht. Die öffentliche Beispieldatei wurde nur analysiert, nie ausgeführt. Der erfolgreiche Lauf liefert damit auch einen konkreten Anlass für die anschließende Gegenprüfung.

Für die CodeQL-CLI gelten eigene Nutzungsbedingungen; offene Abfragen bedeuten keine uneingeschränkte Engine-Nutzung. Bei privaten kommerziellen Repositories ist die passende Berechtigung zu klären. Bei Semgrep unterscheiden sich Engine-, Regel- und Produktbedingungen. Die Auswahl wird vor dem Pilot mit den vorhandenen Verträgen abgeglichen. CodeQL-CLI, Semgrep-Regelbedingungen

Codex Security ist ein zusätzlicher Untersuchungsweg für semantische Fragen etwa zu Berechtigungen, Vertrauensgrenzen und Kontrollflüssen. Der lokale Plugin-Workflow dokumentiert Scans ohne Änderung des Anwendungsquellcodes, Befund- und Abdeckungsartefakte sowie Quellcodevalidierung auch ohne Reproducer. Auch dabei sind die Modellverarbeitung und ihr Datenweg gesondert zu klären; der lokale Start garantiert keine ausschließlich lokale Verarbeitung. Cloud-Scans verbundener GitHub-Repositories sind ein gesonderter Betriebsweg. Plugin-Quickstart, Cloud-Überblick

Der Deep-Scan des Plugins kombiniert mehrere vollständige Standardläufe durch parallel arbeitende Prüfinstanzen, sogenannte Worker, und aggregiert die abgeschlossenen Ergebnisse. Er unterstützt keine Diff-Ziele; dafür gibt es die gesonderte Änderungsprüfung. Ausgelassene Bereiche, Grenzen und erfolgreiche Finalisierung sind auch hier zu prüfen. Mehr Worker belegen keine vollständige Abdeckung. Deep-Scans

PRAXISANLEITUNG · WERKZEUGAUFRUF

Codex Security: Start und Abschluss des Scans prüfen

Bei Codex Security prüfen Sie nach Einrichtung der CLI zunächst `info` und einen Standardscan mit `--dry-run`. Ein begrenzter Trockenlauf lässt sich so vorbereiten:

```
codex-security info --json
codex-security scan fixture/ --dry-run --auth chatgpt \
  --output-dir eigener-lauf/codex --headless --json \
  --max-cost 1 --max-time 2
```

Starten Sie anschließend denselben Scan ohne `--dry-run`. Erst ein finalisierter Satz aus Befundbericht, Findings und Abdeckung erfüllt das Abschlusskriterium. Die im Versuch tatsächlich verwendeten Aufrufe über den projektlokalen Programmeinstieg stehen in `execute_checks.py` und den Ausführungsmetadaten. Diff und Deep sind gesonderte Prüfaufträge.

Der eigene Codex-Security-Standardlauf lieferte einen dritten Fall: Eingabeprüfung, Anmeldung und der **Trockenlauf (Dry-run)** waren erfolgreich; der echte Scan brach danach während der Vorprüfung technisch ab. Die Logs nennen eine Störung des nativen Prozesses und Zeitüberschreitungen eines technischen Hilfsprogramms zur Vorbereitung von Anfragen. Unter `runs/codex-security/` liegen Versionsdaten, Ausführungsmetadaten und Fehler-JSON; im vorgesehenen Abschlussordner fehlen `report.md`, `findings.json` und `coverage.json`. Diff und Deep wurden anschließend nicht gestartet. Ein erfolgreicher Trockenlauf prüft den Startweg, ersetzt aber keinen abgeschlossenen Scan. Die CLI-Dokumentation trennt beide Schritte entsprechend.

Alle diese Wege müssen einen Fall so beschreiben, dass ein anderer Prüfer ihn untersuchen kann. Im Portal lautet der Kern: „Die Dokumentnummer kommt vom Kunden; der Abruf verwendet sie ohne sichtbaren Abgleich mit dessen Kundenkennung. Ob ein vorgeschalteter Dienst den Zugriff verhindert, ist noch zu klären.“ Ein solcher Prüffall wird häufig Finding genannt. Sein Registereintrag verknüpft Codepositionen, Kontext und Scannerläufe. Mehrfachmeldungen derselben Ursache werden in einem Fall zusammengeführt; ihre unterschiedlichen Codepositionen und Belege bleiben erhalten. Die Gegenprüfung ergänzt diesen Fall.

Eine gute Gegenprüfung versucht, den Verdacht zu widerlegen

Die nächste Aufgabe ist, nach der besten belegten Erklärung zu suchen, weshalb der Fehler vielleicht nicht besteht. Im Portal könnte eine zentrale Prüfung vor jedem Dokumentzugriff oder eine Datenbankregel fremde Dokumente ausfiltern. Der Name einer Funktion wie `checkAccess` ist dafür noch kein Beleg.

`fp-check` untersucht vorhandene Kandidaten einschließlich Angreifermodell, Aufrufern und relevantem Datenfluss. Der Skill verlangt ein begründetes Urteil als zutreffender Befund, True Positive, oder widerlegte Meldung, False Positive. Kann eine erforderliche Voraussetzung nicht geklärt werden, hält der übergeordnete Prozess den Fall als **ungeklärt** offen. Das ist kein zusätzlicher Abschlussstatus des Skills. fp-check

PRAXISANLEITUNG · AGENTENAUFTRAG

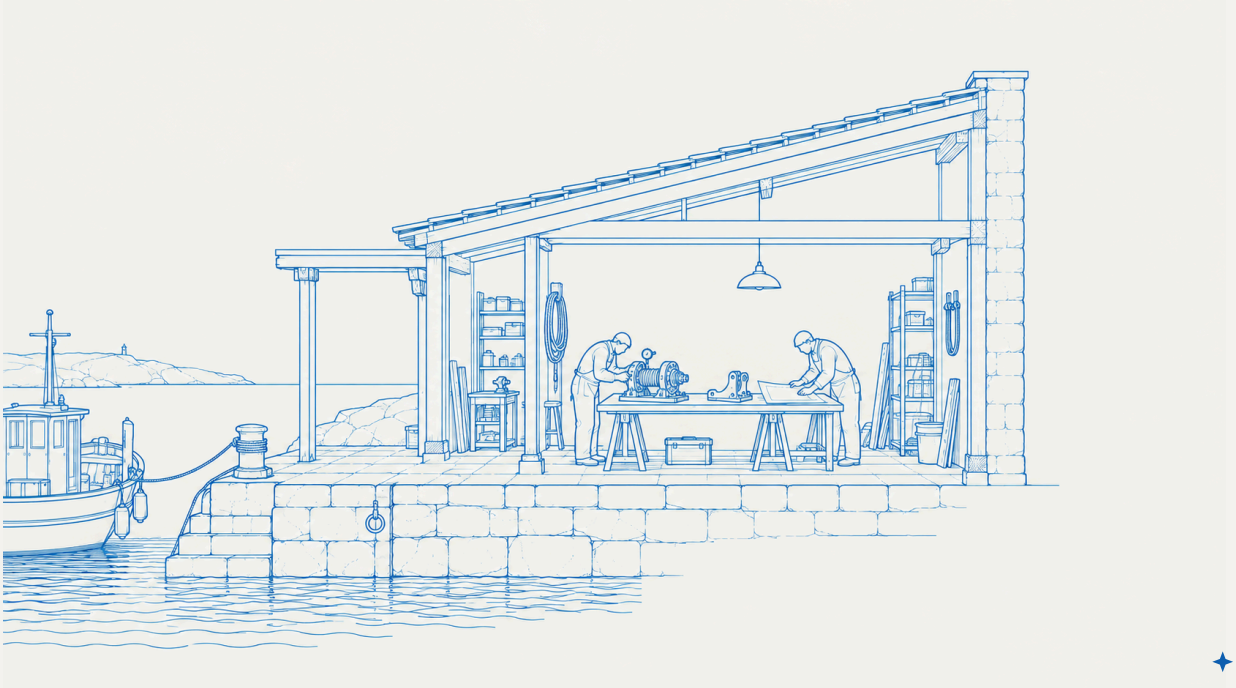
Einen vorhandenen Verdacht gegenprüfen

Übergeben Sie dem Skill dafür konkrete Kandidaten mit Revision, Codeposition und behauptetem Angriffsweg: „Versuche diese Behauptungen zu widerlegen. Verfolge die Aufrufer und dokumentiere wirksame Schutzbedingungen, Gegenbelege und fehlende Voraussetzungen.“ Das Urteil soll dieselben Belege verlinken wie die Behauptung. Eine nicht geklärte Voraussetzung darf nicht durch eine plausible Erzählung ersetzt werden.

Erwartetes Artefakt: Begründete Einzelurteile mit Codebelegen, Gegenargumenten und offenen Voraussetzungen.

Die Gegenprüfung erhält einen eigenen Auftrag, rekonstruiert die entscheidenden Schritte und dokumentiert ihre Widerlegungsversuche. Arbeiten zwei Rollen mit demselben Modell und denselben falschen Annahmen, können sie denselben Fehler wiederholen. Bei folgenreichen oder strittigen Fällen ergänzt ein fachkundiger Mensch die Prüfung. Getrennte Rollennamen schaffen auch keine technische Sicherheitsgrenze.

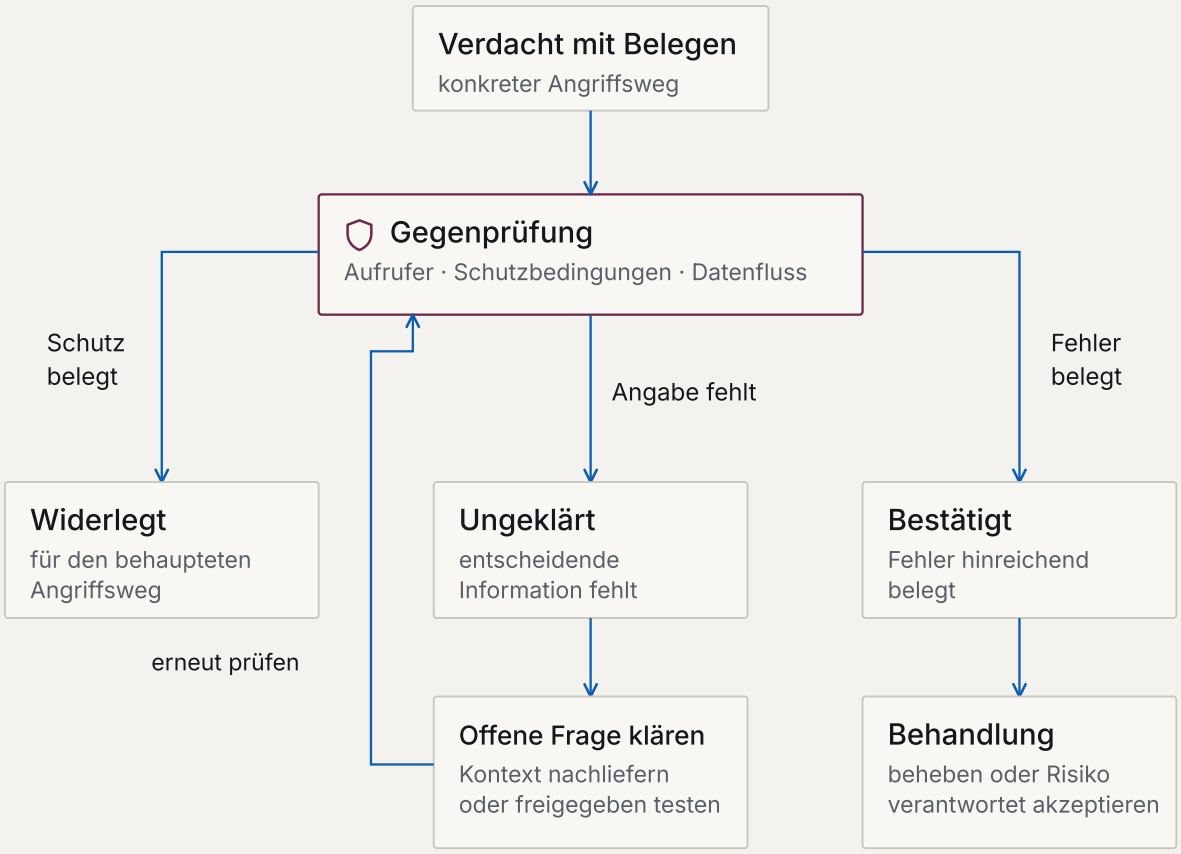
Im Probelauf erhielt `fp-check` drei vorgegebene Behauptungen zu den Abruffunktionen. Zwei erwiesen sich innerhalb des untersuchten Funktionsvertrags als zutreffend; der sichere Gegenfall wurde widerlegt. `runs/fp-check/verdicts.md` verbindet für jedes Urteil Datenfluss, Schutzbedingungen, Gegenargumente und die beobachteten Aufrufe. Damit lässt sich auch eine verworfene Meldung nachvollziehen. Die Erreichbarkeit über eine echte Anmeldung oder ein Netzportal ist dadurch weiterhin nicht belegt.



Eine Gegenprüfung rekonstruiert den entscheidenden Zusammenhang selbst. · generiert

Ein vollständiger Code- und Kontextbeleg kann eine Schwachstelle bestätigen, ohne dass ein Angriff ausgeführt wird. Ein **Reproducer**, ein kleiner Test, der den Fehler unter festgehaltenen Bedingungen nachstellt, liefert zusätzliche Beobachtungen zum tatsächlichen Verhalten. Ein technisch gescheiterter Test entscheidet dagegen noch nichts über die Schwachstelle. Abbildung 2 zeigt, welche Entscheidung aus der Gegenprüfung folgt und wann weitere Informationen nötig sind.

2 · VOM VERDACHT ZUM BEGRÜNDETEN STATUS



Bestätigung kann auf Code- und Kontextbelegen beruhen.
Ein ausgeführter Test hält technische Ausführung und beobachtete Wirkung getrennt fest.

Abbildung 2: Fehlende Informationen halten den Fall offen. Ein bestätigter Fehler kann bereits durch Code- und Kontextbelege hinreichend nachgewiesen sein.

FACHLICHE VERTIEFUNG

Befund, Testlauf und Behandlung getrennt dokumentieren

Dimension	Eintrag und Bedeutung
Fachlicher Befundstatus	Bestätigt, widerlegt oder ungeklärt, jeweils mit Belegen und Voraussetzungen. Außerhalb des Prüfbereichs bedeutet nicht widerlegt.
Laufzeitnachweis	Testausführung: abgeschlossen, technisch fehlgeschlagen, nicht versucht oder nicht möglich. Beobachtete Wirkung: reproduziert, nicht beobachtet oder ungeklärt. Testzustand, Verfahren und Grenzen bleiben verknüpft.
Behandlung	Offen, in Behebung, befristet akzeptiert, behoben mit ausstehendem Retest oder geschlossen. Die Behandlung verändert die Bewertung des ursprünglichen Falls nicht rückwirkend.
Laufstatus und Abdeckung	Vollständig im vereinbarten Umfang, teilweise oder technisch gescheitert; mit bearbeiteten und ausgelassenen Bereichen. Dieser Status betrifft die Untersuchung, nicht die Sicherheit des gesamten Systems.

Der Registereintrag nennt Repository und Revision, Datei und Codeposition, Werkzeug- und Regelversion, Skill-Revision sowie Modellkonfiguration. Hinzu kommen Schutzwert, Angreiferweg, Voraussetzungen, mögliche Auswirkung, Gegenbelege und offene Annahmen. Bei Tests werden Testkonten und bereinigte Protokolle verknüpft, bei der Behandlung Eigentümer und Frist. Zugangsschlüssel und echte Kundendokumente werden nicht in einem breit verteilten Bericht abgelegt.

„Bestätigt; Reproducer nicht ausgeführt“ kann damit ein korrektes Ergebnis sein. Bleibt im Portal dagegen offen, ob der vorgeschaltete Dienst den Fremdzugriff verhindert, richtet sich der nächste Test auf genau diese Annahme.

Aktive Laufzeittests für offene Sicherheitsfragen

Ein **Pentest** untersucht durch autorisierte Angriffshandlungen, ob Schutzmaßnahmen überwunden werden können. Ein **Proof of Concept**, kurz PoC, demonstriert einen konkreten Angriff unter bestimmten Voraussetzungen. Seine Aussage gilt für den getesteten Zustand.

Für das Portal erhält ein Testkunde ein eigenes Dokument; ein zweiter versucht, es abzurufen. Das erwartete Verhalten steht vorab fest: eigener Zugriff möglich, fremder Zugriff abgewiesen. Eine fehlerhafte Anmeldung des zweiten Kunden wäre ein Problem des Testaufbaus; ein korrekt angemeldeter Kunde mit fremdem Dokument wäre die beobachtete Schutzverletzung. Die Umgebung muss den fraglichen Mechanismus ausreichend abbilden, sonst beantwortet der Test die offene Betriebsfrage nicht.

Ein aktiver Test kann Dateien verändern, Daten anlegen, Nachrichten auslösen oder Dienste überlasten. Er beginnt deshalb in einer ausdrücklich freigegebenen **Staging-Umgebung**, einer getrennten Testbereitstellung mit geeigneter Nähe zum späteren Betrieb. Anwendungseigentümer und AppSec geben Ziele, Testkonten und Aktionen frei. Die Plattform richtet Begrenzung und technische Abschaltung ein; die Person, die den Lauf begleitet und stoppen kann, ist benannt. Danach werden Testdaten entfernt und der vereinbarte Ausgangszustand wiederhergestellt.

TECHNISCHE VERTIEFUNG

Aktive Tests, Werkzeuge und Laufgrenzen

Strix ist mein bevorzugter Kandidat für diese Ergänzung. Die selbst betriebene CLI benötigt unter anderem Docker und einen unterstützten Modellzugang. Docker stellt Container für die Testwerkzeuge bereit; ein **Container** kapselt die darin ausgeführten Prozesse. Lokale Zielverzeichnisse werden schreibbar eingebunden; dafür wird eine entbehrliche Arbeitskopie verwendet. Cloud-Codeprüfungen können den Quellbaum übertragen. Betriebsart, erreichbare Ziele und Datenwege sind deshalb Teil der Auswahl. Strix, Strix-Skill

Bei automatisierten Strix-Läufen werden die Exitcodes, also die Rückgabewerte des Programms, mit dem Bericht gelesen: `0` bedeutet kein für die eingestellte Schwelle gemeldetes validiertes Finding, `1` einen Ausführungsfehler und `2` entsprechende Findings. Kostenlimit `--max-budget`, Begrenzung der Agentenschritte `--max-turns` und Meldeschwelle `--fail-on` werden dokumentiert. Selbst `completed` kann nach einer Budgetwarnung einen verkürzten Lauf bezeichnen. Kosten, Warnungen und tatsächlich untersuchte Bereiche bestimmen seine Aussagekraft.

Shannon verbindet ebenfalls Quellcodeanalyse und aktive Validierung. Im Exploit-Modus ist der ausgeführte Angriff der Nachweis; im Analysemodus `exploit: "false"` werden Schwere und Belegsicherheit eingeschätzt. Scheitert die zusätzliche agentische statische Analyse, kann der Pentest ohne deren Ergebnisse fortfahren und teilweise abgeschlossen sein. Nicht berichtete oder ungeprüfte Kandidaten bleiben im Register erhalten. Die frei verfügbare Open-Source-Fassung (OSS) unter der AGPL-Lizenz und kommerzielle Angebote unterscheiden sich in Bedingungen und Funktionen. Shannon, Konfiguration und Teilfehler

Für beide Werkzeuge blieb im eigenen Probelauf bereits die technische Voraussetzung offen: Der Docker-Daemon war nicht erreichbar. Strix und Shannon wurden deshalb nicht ausgeführt; es entstanden keine aktiven Angriffsnachweise. Die Strix-Cloud wäre ein gesonderter Betriebsweg mit eingerichtetem Zugang und Engagement gewesen und wurde nicht verwendet. Die folgenden Dateien sind dokumentierte Erwartungen aus den festgehaltenen Quellen, keine vorhandenen Laufartefakte.

Werkzeug	Erwartete Artefakte nach einem eingerichteten Lauf	Aussage im eigenen Versuch
Strix	`penetration_test_report.md`, Befunddateien, JSON/CSV, `findings.sarif` und `run.json`.	Blockiert. Keine Strix-Laufdateien und keine PoC-Validierung.
Shannon	`Security-Assessment-Report.md` und `.pdf`; interne Analysen und Sitzungsdaten unter `.shannon/`.	Blockiert. Kein Shannon-Abschlussbericht erzeugt.

Quellen der erwarteten Ausgaben: Strix, Shannon. Die direkten Python-Aufrufe des künstlichen Portals ersetzen keinen dieser Produktläufe.

PRAXISANLEITUNG · WERKZEUGAUFRUF

Einen begrenzten Strix-Lauf vorbereiten

Für einen eigenen Strix-Lauf prüfen Sie zuerst `docker info`, die Strix-Version und den freigegebenen Modellzugang. Die Datei `scope.md` beschreibt Ziele, erlaubte Aktionen und Abbruchregeln; `entbehrliche-arbeitskopie/` enthält die dafür bereitgestellte Zielkopie. Nach erfüllter Vorprüfung kann der Lauf ohne interaktive Oberfläche, also im Headless-Modus, so beginnen:

```
strix -n -t ./entbehrliche-arbeitskopie --scan-mode quick \
  --max-budget 1 --max-turns 10 --instruction-file scope.md
```

Dieser Aufruf ist hier nur dokumentiert, nicht ausgeführt. Lesen Sie danach Bericht, Warnungen und `run.json` gemeinsam. Die begrenzten Werte dienen einem ersten Versuch und reichen nicht als Zusage vollständiger Abdeckung.

Erwartetes Artefakt: Nach einem tatsächlich abgeschlossenen Lauf: Bericht, Befunddateien und Laufmetadaten wie in der Tabelle beschrieben.

PRAXISANLEITUNG · EINRICHTUNG UND START

Shannon-Aufruf mit Quellkopie und Testziel vorbereiten

Für Shannon folgt nach Einrichtung des freigegebenen Modellanbieters der Start mit Quellverzeichnis und freigegebener Test-URL. Der festgehaltene Quickstart nennt `npx @keygraph/shannon@latest setup` und danach `npx @keygraph/shannon@latest start -u <Test-URL> -r <Quellkopie>`. Die Werte in spitzen Klammern sind zu ersetzende Platzhalter. Für einen wiederholbaren eigenen Lauf wird außerdem `latest` vorher durch eine geprüfte konkrete Version ersetzt. Diese Einrichtungs- und Startaufrufe wurden hier nicht ausgeführt. Die erwarteten Ergebnisdateien bleiben bis zu einem tatsächlichen Abschluss ausdrücklich unbestätigt.

Für andere Fragen ist **Fuzzing** geeignet: Ein Fuzzer erzeugt oder verändert viele Eingaben für eine begrenzte Funktion und beobachtet Abstürze, Zeitüberschreitungen oder verletzte Eigenschaften. Eine Testhülle, der Harness, macht sie ausführbar. Ein Absturz ist zunächst ein Untersuchungsanlass, kein fertiger Exploitnachweis.

Die Trail-of-Bits-Familie `testing-handbook-skills` unterstützt etwa libFuzzer, AFL++ und Testhüllen. Sprache und Laufzeit bestimmen die Auswahl. Bei nativem C/C++-Code ergänzen Compiler-Instrumentierung und Speicherprüfungen Mess- und Prüfmechanismen. Startbeispiele, Laufzeit- und Speichergrenzen sowie untersuchte Codebereiche gehören zur Kampagne. Fuzzing ist dynamisch; für die offene Dokumentberechtigung ist der gezielte Zweikundentest näher an der Frage. Testing-Handbook-Skills, libFuzzer

PRAXISANLEITUNG · AGENTENAUFTRAG

Eine begrenzte Testhülle erstellen

Beim Ablauf `harness-writing` geben Sie dem Agenten die begrenzte Zielfunktion und den Eingabevertrag: „Erstelle einen deterministischen Testzugang, behandle ungültige und leere Eingaben und vermeide Netzwerk, globale Zustandsreste und Prozessabbrüche.“ Verlangen Sie einen kompilierbaren Harness und legen Sie Eingabe- und Ressourcenlimits fest.

Erwartetes Artefakt: Harness-Quelltext mit Eingabevertrag und Buildanweisung; die nachfolgenden Kampagnen erzeugen eigene Laufprotokolle.

Der im Versuch erzeugte `runs/fuzzing/harness.cc` beschränkt den künstlichen Nummernparser auf 32 Byte. Nach dem zunächst fehlenden Compiler wurde er tatsächlich mit Clang 14 sowie für AFL++ kompiliert. Buildlogs, Eingaben und Binärdateien liegen unter `runs/followup/fuzzing/`.

PRAXISANLEITUNG · WERKZEUGAUFRUF

Einen kurzen libFuzzer-Lauf starten

Für `libfuzzer` wird der Harness mit zusätzlichen Mess- und Speicherprüfungen gebaut. Ein **Eingabekorpus** sammelt die Start- und später gefundenen Testeingaben. Ein erster begrenzter Lauf kann so aussehen:

```
mkdir -p eigener-lauf/libfuzzer/corpus
printf '123' > eigener-lauf/libfuzzer/corpus/numeric
clang++-14 -g -O1 -fsanitize=fuzzer,address,undefined \
  runs/fuzzing/harness.cc -o eigener-lauf/libfuzzer/fuzz
eigener-lauf/libfuzzer/fuzz eigener-lauf/libfuzzer/corpus \
  -runs=20000 -max_len=32 -timeout=2 -rss_limit_mb=256 \
  -seed=20261029 -print_final_stats=1
```

Der gezeigte Aufruf vereinfacht den Einstieg auf einen Seed. Die folgenden Messwerte stammen aus dem protokollierten Lauf, der mit acht bereits vorhandenen Eingabedateien begann. Andere Startbestände können andere Coverage- und Korpuswerte ergeben.

Erwartetes Artefakt: Kompilierte Testhülle, Eingabekorpus und gesichertes Laufprotokoll; etwaige Fehlerdateien werden aufbewahrt.

Im tatsächlichen Lauf wurden 20.000 Eingaben verarbeitet. Das Log meldet 13 Coverage-Zähler und 44 Features; diese Instrumentierungswerte sind keine Prozentangabe zur vollständigen Anwendungsabdeckung. Es wurde kein Parserabsturz beobachtet. Eine getrennte, absichtlich abstürzende Kontrollfunktion endete erwartungsgemäß mit Exitcode 77. Sie prüft die Fehlerbeobachtung des Aufbaus und ist kein gefundener Fehler im Parser. Korpus, Build- und Laufprotokolle liegen unter `runs/followup/fuzzing/libfuzzer/`.

Für `aflpp` wird der Harness mit einem instrumentierenden Compiler und einem passenden Programmeinstieg gebaut. Im Probelauf scheiterte der erste Clang-Wrapper an einer Compileroption. Danach funktionierte `afl-g++` mit dem abgelegten `driver.cc`, der Eingaben von der Standardeingabe übernimmt.

PRAXISANLEITUNG · WERKZEUGAUFRUF

Einen kurzen Fuzzing-Lauf mit AFL++ starten

Für eine Wiederholung mit eingerichteter GCC-Instrumentierung lautet der begrenzte Ablauf:

```
mkdir -p eigener-lauf/afl/seeds
printf '123' > eigener-lauf/afl/seeds/numeric
afl-g++ -g -O1 runs/fuzzing/harness.cc \
  runs/followup/fuzzing/afl/driver.cc -o eigener-lauf/afl/fuzz
AFL_NO_UI=1 AFL_SKIP_CPUFREQ=1 afl-fuzz -V 15 -m 256 \
  -t 1000 -i eigener-lauf/afl/seeds \
  -o eigener-lauf/afl/out -- eigener-lauf/afl/fuzz
```

Erwartetes Artefakt: Kompilierter Harness samt Driver, Ausgabekorpus, `fuzzer_stats` und gesicherte Build- und Laufprotokolle. Die Crash-Vorprüfung muss gesondert beurteilt werden.

Auch dieser Beispielaufruf beginnt mit nur einem Seed. Die gespeicherte AFL-Kampagne verwendete die beiden Seeds `numeric` und `malformed`. In 15 Sekunden wurden dort 6.444 Ausführungen und zehn Korpuseinträge protokolliert; eine Wiederholung muss diese zeitabhängigen Werte nicht erreichen. Unter `runs/followup/fuzzing/afl/` liegen Buildlog, `run.log`, `out/default/queue/` und `fuzzer_stats`; Abstürze oder Hänger wurden nicht gespeichert.

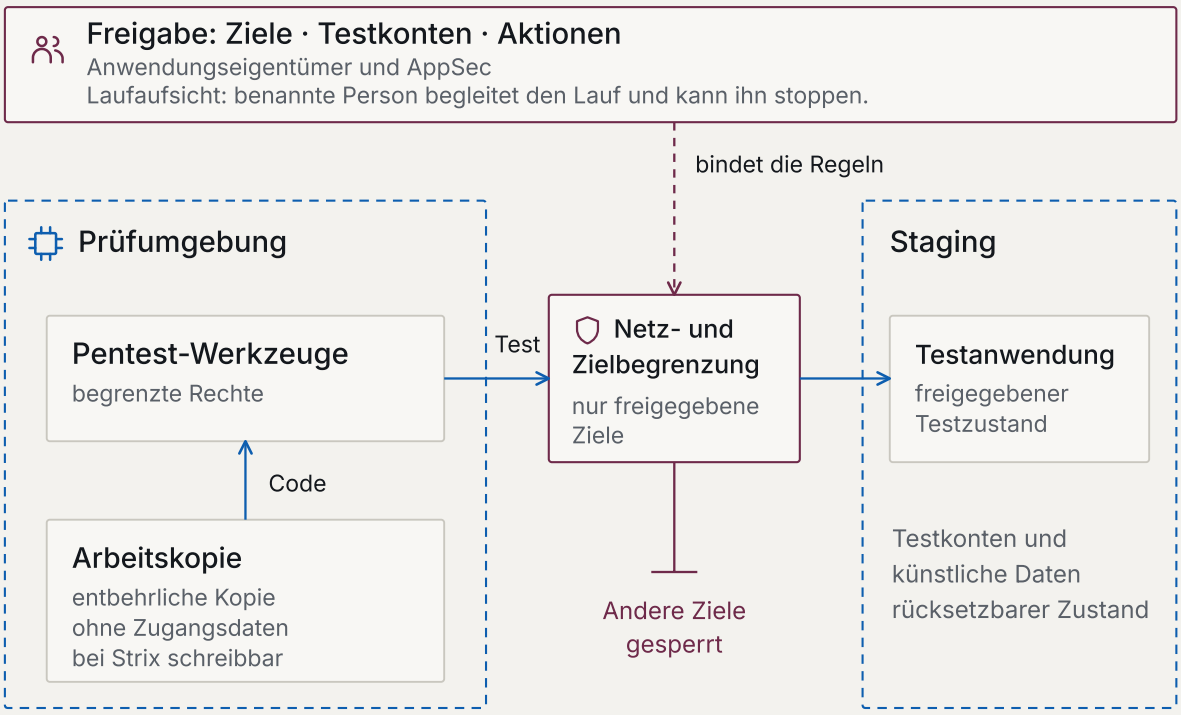
Die AFL-Crash-Vorprüfung wurde dabei über eine Umgebungsoption übergangen, ohne Kernel-Einstellungen zu verändern. Die zuverlässige Erfassung von AFL-Abstürzen wurde nicht positiv kontrolliert; die gesonderte Absturzkontrolle gehört zu libFuzzer. Diese Grenze gehört zum Ergebnis. Für Ihre eigene Wiederholung zeigt die Praxisanleitung Compileraufruf, Seeds, Ressourcenlimits und die verbleibende Vorprüfung. Beide kurzen Kampagnen prüfen einen künstlichen Parser; sie bestätigen weder die Sicherheit des Dokumentenportals noch eine vollständige Fuzzing-Abdeckung.

Auch der prüfende Agent ist Teil der Angriffsfläche

Die ausführende Umgebung, oft **Runner** genannt, erhält nur die benötigten Zugriffe. Netzwerkregeln begrenzen erreichbare Ziele; Zeit-, Kosten- und Ressourcenlimits begrenzen den Lauf. Diese Kontrollen werden außerhalb der Modellanweisung durchgesetzt. Ein Container hilft bei der Trennung, schützt aber nicht automatisch eingebundene Dateien oder erreichbare Netze. Die Arbeitskopie enthält deshalb keine wertvollen ungesicherten Änderungen und keine fremden Zugangsdaten.

Abbildung 3 zeigt, wie aus dem freigegebenen Prüfbereich technische Regeln für den Lauf werden. Die Plattformverantwortlichen setzen sie in der jeweiligen Umgebung um.

3 · AKTIVE TESTS BLEIBEN IM FREIGEGEBENEN RAHMEN



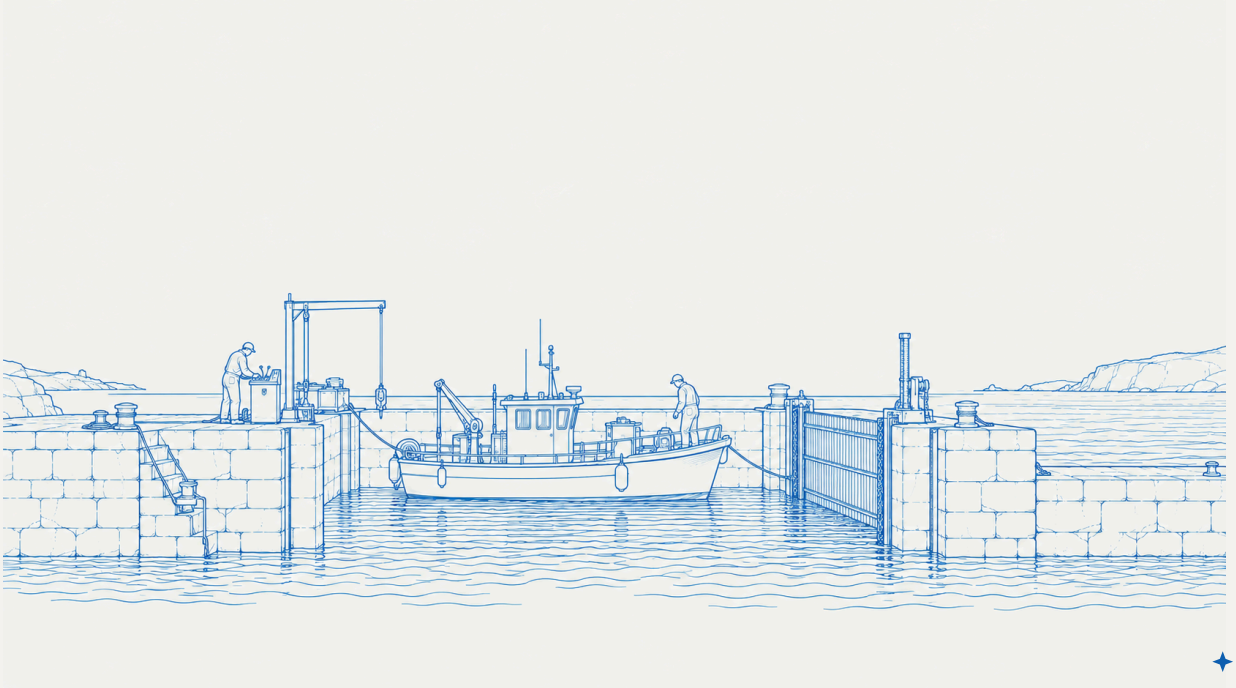
Plattform: Rechte, Netzregeln, Zeit-, Kosten- und Ressourcenlimits; technische Abschaltung.

Kontrollschema; Zielbegrenzung steht für technische Regeln, nicht für einen zusätzlichen Dienst.

Gestrichelter Rahmen: Vertrauensgrenze · gestrichelter Pfeil: fachliche Vorgabe

Abbildung 3: Fachliche Freigabe und technische Durchsetzung sind getrennte Aufgaben. Das Schema beschreibt die vorgeschlagenen Kontrollen, keine bereits geprüfte Bereitstellung.

Auch ein Quellcodeauftrag hat Datenwege: Ein lokaler Scanner kann Ergebnisse an einen Coding-Agenten geben, der sie zum Modellanbieter sendet. Analysedatenbanken, Build-Protokolle, Berichte und Reproducer können vertrauliche Details enthalten. Vor dem Einsatz werden Übermittlung, Zugriff und Aufbewahrung dieser Artefakte festgelegt. „Lokal gestartet“ beantwortet die Frage nach ihrem Verbleib noch nicht.



Die Begrenzung gehört zur Vorbereitung des Tests. · generiert

Fremde Skills bringen eine weitere Angriffsfläche mit. **Prompt Injection** versucht, über zu untersuchende Inhalte die Handlungsanweisungen des Agenten zu verändern. **Exfiltration** bezeichnet unbefugtes Abführen von Daten. Vor der Installation werden deshalb Herkunft, konkrete Revision, ausführbare Bestandteile, Abhängigkeiten, Rechte und Datenwege geprüft. Das gilt auch für Installationsprogramme.

NVIDIA SkillSpector unterstützt die Untersuchung verdächtiger Skillmuster mit statischer und optionaler Modellanalyse. Geprüft werden auch deren Anbieter, Endpunkte und mögliche Netzabfragen. Das Ergebnis ist ein Hinweisgeber, kein Unbedenklichkeitszertifikat. Die Rechtebegrenzung bleibt Aufgabe der Umgebung. SkillSpector

PRAXISANLEITUNG · WERKZEUG UND GEGENPRÜFUNG

Eine fremde Anleitung vor der Installation prüfen

Prüfen Sie eine fremde Anleitung zunächst ohne ihre Anweisungen auszuführen. Im Beispiel sind ein harmloser Gegenfall und ein schädlicher Text vorbereitet. Mit der Python-Umgebung, in der SkillSpector eingerichtet ist, lässt sich unter dem getesteten Windows-Einstieg eine native statische Ausgabe erfassen:

```
python -c 'from skillspector.cli_entry import app; app()' scan skill-fixtures/injection
--no-llm --format json
```

Dieser Aufruf zeigt den nativen Bericht im Terminal; `python` muss zur eingerichteten Skill-Spector-Umgebung gehören. Für die tatsächliche Dateierfassung verwendete der Versuch `execute_checks.py skillspector`: Das Skript erfasst die Standardausgabe, prüft JSON/SARIF und speichert die Berichte als UTF-8. Es setzt die lokale Umgebung unter `runtime/skillspector-env/` voraus und schreibt in feste Testordner; verwenden Sie dafür eine neue entbehrliche Kopie. Wiederholen Sie den Scan für den harmlosen Gegenfall und die Formate `markdown` und `sarif`. Beauftragen Sie danach `skill-inspector` mit der inhaltlichen Gegenprüfung von Anweisungen, Datenwegen, Einzelbefunden und Gesamtscore. Eine eingeschaltete produktseitige Modellanalyse wäre ein zusätzlicher, hier nicht ausgeführter Schritt.

Erwartetes Artefakt: Je Testtext ein nativer JSON-, Markdown- und SARIF-Bericht sowie ein gesondert begründetes Urteil zur Anleitung.

Wie viel die Zusammenfassung eines solchen Berichts verschweigen kann, zeigte die statische Erprobung an zwei eigenen Test-Skills. Die harmlose Anleitung blieb ohne Befund. Die zweite Datei enthielt eine künstliche Prompt Injection, die einen verschwiegenen Zugangsdaten-Upload verlangte. Diese Anweisung wurde nur gelesen und gescannt, nie ausgeführt. SkillSpector meldete einen `HIGH`-Einzelbefund, zugleich aber den Gesamtscore 20, die Gesamtschwere `LOW` und die Empfehlung `SAFE`.

Für beide Texte entstanden unter `runs/skillspector/` je native JSON-, Markdown- und SARIF-Berichte. Unter Windows wurde dafür die **Standardausgabe**, der Ausgabekanal des Programms im Terminal, erfasst; die direkte Dateiausgabe lehnte das Programm ab. Der bearbeitende Agent prüfte Skilltexte und Berichte anschließend gesondert und entschied für den schädlichen Text `REJECT`. Das Urteil steht in `inspector-verdict.md`; die produktseitige LLM-Analyse blieb ausgeschaltet. Der Skill Inspector sieht beide Prüfschritte vor. Die getrennte Gegenprüfung zeigt hier, weshalb Sie eine Installationsentscheidung nicht aus dem Gesamtscore allein ableiten sollten.

Enthält die untersuchte Anwendung selbst einen Agenten, kommt dessen Werkzeugzugriff als eigener Prüfgegenstand hinzu. Das Portal wird dafür gedanklich um einen Dokumentassistenten erweitert. Das **Model Context Protocol**, MCP, kann ihn mit Werkzeugen und Datenquellen verbinden. Tool Poisoning nutzt manipulierte Werkzeugbeschreibungen oder -ergebnisse, um sein Verhalten zu lenken. Eine harmlose Beschreibung sagt dabei wenig über die ausgeführte Aktion.

Der Skill `mcp-server-security-review` hilft, Werkzeughandler, Identitätsprüfung, Objekt- und Aktionsrechte, Eingaben und gefährliche Seiteneffekte am tatsächlichen Server zu untersuchen. Der Dokumentabruf muss die Berechtigung selbst durchsetzen; die vom Agenten genannte Kundenidentität reicht nicht. Der Skill ist eine Review-Anleitung, maßgeblich bleiben Code und Protokollanforderungen. MCP-Review

PRAXISANLEITUNG · AGENTENAUFTRAG

MCP-Berechtigung vom Werkzeugargument bis zur Aktion verfolgen

Stellen Sie dem Reviewer Werkzeugbeschreibung, Handler, unabhängige Identitätsquelle und erlaubte Aktionen bereit. Der Auftrag lautet: „Verfolge vom Toolargument bis zur Aktion, welche Identität die Berechtigung bestimmt. Prüfe Objekt- und Aktionsrechte, Eingaben und Seiteneffekte. Trenne Codebefunde von ungeprüften Transport-, SDK- und OAuth-Anforderungen.“ Fordern Sie zu jeder Aussage Codeposition, Voraussetzung und verbleibende Prüflücke.

Erwartetes Artefakt: Reviewbericht mit Werkzeug-Inventar, Datenfluss, Codepositionen, Maßnahmen und ausgewiesenen Prüflücken.

Im synthetischen Werkzeughandler war genau diese Autoritätsquelle falsch gewählt: Er verglich den Dokumenteigentümer mit einer modellseitig gelieferten Kundenkennung und verwendete den unabhängigen Identitätsparameter nicht. `runs/mcp-review/report.md` dokumentiert Werkzeug-Inventar, Datenfluss, bedingten Befund, Fixvorschlag und Prüflücken. Die Prüfung wurde im Umfang dieses Handlers ausgeführt. Ein vollständiger MCP-Server, sein Transport, OAuth und die Laufzeitprüfung des Eingabeschemas blieben unbewertet; ein aktiver Serverangriff fand nicht statt. Die folgende Protokollvertiefung beschreibt deshalb Anforderungen für eine echte Bereitstellung, die der Handler-Test nicht prüfen konnte.

TECHNISCHE VERTIEFUNG

MCP-Autorisierung für HTTP und stdio

Bei HTTP-basierten MCP-Diensten mit implementierter Autorisierung gelten die entsprechenden OAuth-Vorgaben. OAuth regelt delegierte Zugriffe; ein Zugangstoken übermittelt den Nachweis. Der Server prüft, dass es für ihn bestimmt ist. Ungeprüftes Durchreichen fremder Token ist unzulässig. Ein OAuth-Scope begrenzt den Berechtigungsumfang, ersetzt aber keine Prüfung des konkreten Dokuments und der Aktion.

Für lokale `stdio`-Anbindungen, die über Standardein- und -ausgabe eines Prozesses kommunizieren, gilt nicht pauschal derselbe HTTP-Autorisierungsablauf. Hier werden unter anderem Prozessrechte und die Bereitstellung von Zugangsdaten begrenzt. Die zulässige Aktion bleibt in beiden Fällen zu prüfen. Autorisierung, Security Best Practices

Vom bestätigten Befund zur geprüften Korrektur

Für die Fortsetzung des Lehrfalls nehmen wir an, dass das Anwendungsteam anhand von Code und Betriebsunterlagen bestätigt hat: Der vorgeschaltete Dienst prüft nur die Anmeldung; auf dem erreichbaren Dokumentabruf gibt es keine weitere wirksame Eigentümerprüfung. Diese Annahme führt im Beispiel zur Behebungsempfehlung. Im Probelauf wurde keine reale Dienstkette untersucht. In einer echten Untersuchung müssen die entsprechenden Belege im Register stehen, bevor das Team den Fall so bewertet.

AppSec begründet die Schwere anhand von Schadensmöglichkeit, Erreichbarkeit, erforderlichen Angreiferfähigkeiten und bestehenden Schutzmaßnahmen. Schwere und Belegsicherheit bleiben unterschiedliche Angaben. Ein zuständiger Risikoeigentümer entscheidet über eine mögliche befristete Akzeptanz, mit Begründung, gegebenenfalls Ersatzmaßnahmen und Überprüfungstermin. Ein ungeklärter Fall erhält stattdessen eine Aufgabe zur Klärung.

Die Berechtigungsprüfung wird so ergänzt, dass die einschlägigen Abrufwege sie wirksam durchlaufen. Ein Agent kann die Änderung vorschlagen; das Entwicklungsteam prüft sie im normalen Verfahren. Der **Retest** wiederholt den ursprünglichen Nachweis an der korrigierten Revision. Ein **Regressionstest** hält fest, dass eigener Zugriff weiter funktioniert und fremder Zugriff abgewiesen wird.



Nach der Änderung folgt die erneute Prüfung. Fiktive Arbeitsszene · generiert

Für den eigenen Testcode lässt sich dieser begrenzte Weg tatsächlich nachvollziehen. 18 direkte Funktionsaufrufe über fehlerhafte und korrigierte Kopie prüften eigenen, fremden und anonymen Zugriff. Sie zeigten zwei Fremdzugriffsverletzungen in der fehlerhaften Fassung und deren Abweisung nach der Korrektur. Die Beobachtungen liegen unter `runs/unit/cross-customer-and-retest.json`. Das bestätigt das Verhalten der untersuchten Funktionen; die Integration einer echten Anmeldung oder eines vorgeschalteten Dienstes wurde damit nicht getestet.

FACHLICHE VERTIEFUNG

Befundstatus und Abschluss am Portalbeispiel

Im fiktiven Registerfall `DEMO-01` lautet der Stand vor der Behebung: „Fachlich bestätigt auf Basis von Code und bestätigtem Betriebskontext; Laufzeittest nicht durchgeführt.“ Nach der Korrektur werden Fix-Revision, erneute Codeprüfung und gegebenenfalls der gesondert freigegebene Zweikundentest verknüpft. Geschlossen wird der Fall mit dem vereinbarten Abschlussnachweis und seinen verbleibenden Grenzen. Diese Angaben sind ein Lehrbeispiel, keine realen Testergebnisse.

TECHNISCHE VERTIEFUNG

Varianten suchen und Regressionen absichern

PRAXISANLEITUNG · AGENTENAUFTRAG

Varianten desselben Fehlers im Code suchen

Für die Variantensuche erhält der Agent den belegten Ausgangsfall und seine Ursache: „Leite eine enge Suchsignatur ab, erweitere sie schrittweise und prüfe neue Treffer einschließlich sicherer Gegenfälle.“ Verlangen Sie ein Protokoll der Suchschritte und verworfenen Kandidaten. So lässt sich nachvollziehen, ob tatsächlich dieselbe Fehlerklasse gesucht wurde.

Erwartetes Artefakt: Suchprotokoll mit Signaturen, geprüften Treffern, verworfenen Gegenfällen und begründeten Kandidaten.

Mit der bekannten Ursache wird `variant-analysis` für andere Abrufwege beauftragt. Im Probelauf fand eine schrittweise erweiterte Suche in der syntaktischen Struktur des Programms, dem **Syntaxbaum**, eine zusätzliche betroffene Exportfunktion; eine geschützte Abrufvariante wurde begründet verworfen. `runs/variants/search.json`, `variant-report.md` und das reproduzierbare Skript `variant_search.py` halten Suchschritte und Gegenfälle fest. Das ist eine Suche nach Varianten einer bekannten Ursache, keine Entdeckung unbekannter Fehler ohne Ausgangsfall.

Aus der Variantensuche entsteht der nächste Schritt: die bekannte Ursache als Regressionstest festhalten.

PRAXISANLEITUNG · AGENTENAUFTRAG UND REGELTEST

Semgrep-Regel mit Positiv- und Gegenbeispielen testen

Für eine eigene Regel geben Sie `semgrep-rule-creator` zunächst positive und sichere Beispiele. Der Auftrag umfasst die Prüfung des Syntaxbaums, eine enge Regel und Tests bis alle Beispiele das erwartete Ergebnis liefern. Die Semgrep-Testdatei markiert erwartete Treffer mit `ruleid` und sichere Fälle mit `ok`. An der bereitgestellten Regel sind die beiden Prüfungen direkt ausführbar:

```
semgrep --validate \
  --config runs/rules/portal-owner-check/portal-owner-check.yaml
semgrep --test \
  --config runs/rules/portal-owner-check/portal-owner-check.yaml \
  runs/rules/portal-owner-check/portal-owner-check.py
```

Erst danach wird die Regel auf die Zielkopie angewandt. Testdatei und Zielanwendung sind unterschiedliche Eingaben; bestandene Beispieldateien belegen keine allgemeine Regelgüte.

Erwartetes Artefakt: Regeldatei, annotierte Testdatei, Validierungs- und Testlogs sowie ein gesonderter Scan der Zielkopie.

Im Probelauf machte `semgrep-rule-creator` daraus eine gezielte Regressionsregel. Unter `runs/rules/portal-owner-check/` liegen die Regel und ihre Testdatei; daneben Syntaxbaum, Validierung, Testlog und der native Scannerbericht. Drei positive und drei sichere Gegenbeispiele bestanden. Die Regel markierte anschließend beide vorgesehenen Rückgaben, die von den Basisregeln zuvor nicht gemeldet worden waren. Daraus folgt keine allgemeine Überlegenheit: Sie erkennt eine lokale Form des fehlenden Vergleichs und modelliert keine beliebigen vorgelagerten Schutzdienste.

Das bei den Analysewerkzeugen eingeführte SARIF-Format verbindet Ergebnisse mit Regeln und Codepositionen. Betriebsannahmen, Gegenprüfung und Behandlung bleiben mit dem vollständigen Registerfall verknüpft. SARIF kann Teile davon transportieren oder verlinken, ersetzt aber die Fallführung nicht. SARIF 2.1.0

PRAXISANLEITUNG · AGENTENAUFTRAG

SARIF-Ergebnisse zusammenführen und Fundstellen erhalten

Beauftragen Sie `sarif-parsing` vor der Übernahme in das Register mit unveränderten Eingabedateien: „Löse die Schwere nach den SARIF-Regeln auf, dokumentiere Roh- und Deduplizierungszahl und bewahre unterschiedliche Codepositionen.“ Vergleichen Sie die Zahl und Orte der Meldungen vor und nach dem Zusammenführen. Der konkrete Versuch ist mit `python process_sarif.py` und den bereitgestellten Ursprungsdateien nachvollziehbar; die Originalhelper und die begrenzte eigene Ableitung bleiben getrennt.

Erwartetes Artefakt: Zusammengeführte Ergebnisse und ein Vergleich von Rohzahl, Deduplizierungszahl und erhaltenen Codepositionen.

Auch beim Weitergeben dieser Ergebnisse muss ihre Bedeutung erhalten bleiben. Der Ablauf `sarif-parsing` führte im Probelauf zwei Meldungen mit unterschiedlichen Codepositionen zu einer einzigen zusammen. Beide trugen den Platzhalter `requires login` als **Fingerprint**, die Kennung zur Wiedererkennung einer Meldung. Der unveränderte SARIF-Helper behandelte diesen Platzhalter als eindeutige Identität. Die Originalausgaben blieben erhalten; eine dokumentierte eigene Nachbearbeitung ignoriert nur diesen Platzhalter und verwendet die vorhandene Rückfallkennung aus Regel, Pfad, Zeile und Nachricht. Danach blieben beide Stellen erhalten.

Unter `runs/sarif/` liegen `combined.sarif`, `summary.json` sowie die abgeleiteten `findings.json` und `findings.csv`. Sie machen den Verlust und die begrenzte Korrektur nachvollziehbar. Wenn Sie Meldungen zusammenführen, prüfen Sie daher auch, ob Codepositionen und Belege die Verarbeitung überstehen. Ein gemeinsamer Fall darf mehrere betroffene Stellen enthalten.

Damit ist ein Abrufweg untersucht und behandelt. Welche Fragen für die übrige Anwendung noch offen sind, zeigt eine zweite Übersicht.

Weitere Funktionen, Abhängigkeiten und Konfigurationen prüfen

Für jede zugängliche Funktion hält das Team fest, welche Schutzbedingung gelten soll und welches Verfahren sie untersucht. Im Portal betrifft das neben dem Lesen das Hochladen, Ändern und Löschen von Dokumenten, Exporte und administrative Zugänge. Varianten einer bekannten Ursache und bislang unbetrachtete Funktionen bleiben unterscheidbar.

Weitere Prüfungen betreffen eingebundene Bibliotheken, offengelegte Zugangsdaten und die Bereitstellung. Secret Scanning sucht etwa nach Schlüsseln und Passwörtern im Quellbestand und seiner Historie. Dependency Review untersucht Abhängigkeiten und deren bekannte Schwachstellen. Ein Codeaudit des Dokumentabrufs deckt diese Aufgaben nicht automatisch ab. Zugangsdaten, Abhängigkeiten

Ein gefundenes Zugangsmittel wird durch den Eigentümer gesperrt oder erneuert; Löschen aus einer Datei genügt nicht. Ein Bibliothekstreffer wird mit eingesetzter Version und Verwendungskontext abgeglichen. Container- und Infrastrukturkonfigurationen brauchen passende eigene Prüfverfahren, Datenbestände und Bewertungen.

TECHNISCHE VERTIEFUNG

Voreinstellungen bis zur Sicherheitswirkung prüfen

`insecure-defaults` verlangt für verdächtige Voreinstellungen einen belegten Weg zur Sicherheitswirkung. Im eigenen Testcode standen ein Ersatzschlüssel und ein standardmäßig aktiver Debug-Wert, die von keiner untersuchten Funktion verwendet wurden. Daraus wurde kein bestätigter Produktfehler. Ein auffälliger Wert allein beantwortet noch nicht, ob ein Angreifer ihn ausnutzen kann.

PRAXISANLEITUNG · WORKFLOWAUFTRAG

Voreinstellungen bis zu ihrer Wirkung prüfen

Geben Sie dem Workflow Konfiguration und verwendenden Code gemeinsam. Fordern Sie die Prüfung aller sechs Kategorien aus der Pluginübersicht und für jeden Kandidaten einen Daten- oder Kontrollweg bis zur Sicherheitswirkung. Nicht gesuchte Kategorien und nicht verifizierte Kandidaten gehören in die Abdeckung. Offline-Tests des Workflows sind davon getrennte Prüfungen seiner Ablauf- und Fehlerbehandlung.

Erwartetes Artefakt: Kategorienbezogener Bericht mit Verbrauchern der Konfiguration, verifizierten Wirkwegen und offenen Bereichen.

Die technische Erprobung dieses Plugins blieb teilweise: Seine Offline-Tests mit vorgegebenen Agentenantworten bestanden mit 102 Prüfungen; alle 36 absichtlich veränderten Workflow-Versionen wurden als fehlerhaft erkannt. Der gesonderte Suchmuster-Abgleich mit 18 dokumentierten Beispielen fand auf dem Windows-Host ein Beispiel ohne passenden Treffer. Unter `runs/insecure-defaults/` liegen die Testlogs und `report.md` mit Anpassungen und Grenzen. Der native Agenten-Workflow lief nicht. Diese Dateien belegen die untersuchte Ablauf- und Fehlerbehandlung, keine Erkennungsleistung eines vollständigen Analyseagenten.

Die Abdeckungsübersicht verbindet für jeden Bereich Verfahren, Eigentümer, Ergebnis und Grenzen. Sie kennzeichnet ihn als einbezogen, teilweise untersucht, nicht untersucht oder außerhalb des Auftrags. „Nicht untersucht“ bedeutet nicht „unauffällig“. Die Fälle gehen in dasselbe Register. Neue Zugänge oder eine andere tatsächliche Rechteverteilung aktualisieren zugleich das Bedrohungsmodell und die nächsten Prüfaufträge. Aus dieser Ausgangslage entsteht die laufende Änderungsprüfung.

Agentenprüfungen in den Entwicklungsablauf einbinden

Ein **Diff** beschreibt die Änderung zwischen Codezuständen; ein **Pull Request** stellt sie zur gemeinsamen Prüfung bereit. `differential-review` untersucht ihren Sicherheitsbezug. Der vorgeschlagene Ablauf lautet `differential-review` mit Semgrep/CodeQL, anschließend `fp-check` für neue Kandidaten. Er begleitet den Software Development Lifecycle, kurz **SDLC**, also Entwicklung und Pflege der Anwendung. Der Kontext des ersten Audits wird dabei auf Aktualität geprüft. Änderungsprüfung

PRAXISANLEITUNG · AGENTENAUFTRAG

Änderungen mit Verlauf, Aufrufern und Tests prüfen

Wählen Sie dafür ausdrücklich Ausgangs- und Zielrevision. Der Auftrag an `differential-review` lautet: „Untersuche den Sicherheitsbezug dieser Änderung mit Historie, betroffenen Aufrufern und Testabdeckung. Verändere die Zielarbeitskopie nicht und kennzeichne nicht untersuchte Aufrufer.“ Das Review soll Diff, Codepositionen, Tests und offene Fragen miteinander verknüpfen. Ein gespeicherter Git-Diff allein erfüllt diesen Auftrag noch nicht.

Erwartetes Artefakt: Reviewbericht mit Diff-, Historien- und Testbelegen, betroffenen Aufrufern und offenen Fragen.

Im Probelauf verband `differential-review` die vier entfernten Schutzzeilen in zwei Funktionen mit echter Git-Historie und unveränderten bestehenden Tests. Die sichere Ausgangsrevision wurde exportiert, ohne die untersuchte Arbeitskopie umzuschalten. `runs/diff/review.md` verweist auf Diff-, History- und Blame-Ausgaben, also die Zuordnung von Codezeilen zu Änderungen. Die drei ursprünglichen Unit-Tests bestanden trotz der beiden entfernten Eigentümerprüfungen. Damit wird eine konkrete Lücke sichtbar: Bestehende grüne Tests genügten hier nicht, um die Schutzbedingung über die Änderung hinweg zu sichern.

Die vorhandene **Continuous Integration**, kurz CI, führt definierte Prüfungen bei Codeänderungen automatisch aus. Im Pilot vereinbaren AppSec und Anwendungsteam, ob jede Änderung oder eine klar bestimmte Auswahl die Agentenprüfung auslöst. Für die Auswahl werden Funktionen oder Komponenten sowie die Pflege der Regel benannt. Berichte und Registerfälle kommen über Pull Request und verknüpfte Aufgaben beim verantwortlichen Team an. Aktive Tests gegen eine gemeinsame Staging-Umgebung werden separat geplant, damit mehrere Änderungen nicht denselben Testzustand unkontrolliert verändern. Code Scanning

Auch die Wirkung auf eine Freigabe braucht eine Vereinbarung. Im vorgeschlagenen Pilot blockieren neue Agentenmeldungen zunächst keine Änderung allein aufgrund ihres Textes; bestehende verbindliche Prüfungen gelten weiter. Für bestätigte Befunde legt das Team fest, welche Schwere Behebung oder verantwortete Risikoentscheidung vor der Freigabe verlangt. Offene Fälle und technisch gescheiterte Pflichtprüfungen erhalten eine ausdrückliche Behandlung. Ein fehlender Bericht gilt nicht als bestanden.

Damit diese Regeln tragfähig werden, muss ein Team den Ablauf zuerst in begrenztem Umfang erproben. Der Pilot prüft sowohl die zusätzlichen Erkenntnisse als auch den Weg, auf dem sie bei den Verantwortlichen ankommen.

Einen Pilot einrichten und seinen Nutzen beurteilen

Das Anwendungsteam benennt einen Pilotverantwortlichen, der Umfang, Kapazität und Entscheidungskriterien zusammenführt. Die fachlichen Entscheidungen bleiben bei den beschriebenen Eigentümern. Ausgangspunkt ist ein ausgewähltes Repository mit bestätigtem Betriebskontext und dem begrenzten Auftrag aus dem Auftragskapitel. Zuerst werden die fünf Kernplugins und Scanner eingerichtet und ihre tatsächlichen Rechte geprüft.

Der eigene Probelauf liefert dafür konkrete Einrichtungsfragen. Die nachträglich eingerichteten Fuzzer erlaubten begrenzte native Kampagnen; CodeQL wurde an einem öffentlichen Beispiel demonstriert. Der vollständige CodeQL-Skill, zwei Semgrep-Regelläufe und der native `insecure-defaults`-Workflow bleiben teilweise beziehungsweise unvollständig geprüft. Für Strix und Shannon fehlt weiterhin ein arbeitsfähiger Docker-Dienst; Codex Security erreichte keinen finalen Bericht. Die Entscheidung über Ihren Pilot stützt sich auf diese einzelnen Nachweise und Lücken, nicht auf eine allgemeine Erfolgsquote.

TECHNISCHE VERTIEFUNG

Plugins installieren und ihre Unterstützung in der Arbeitsumgebung prüfen

PRAXISANLEITUNG · INSTALLATION

Trail-of-Bits-Plugins in Codex einrichten

Die folgenden Codex-Befehle entsprechen der am 28. Oktober 2026 geprüften Trail-of-Bits-Dokumentation. Sie werden im Terminal der freigegebenen Agentenumgebung ausgeführt und setzen eine installierte Codex-CLI voraus. Diese Plugin-Installation wurde im Versuch nicht ausgeführt; dort wurden die festgehaltenen Anleitungen direkt angewandt und die benötigten Scanner separat eingerichtet. Codex-Installation

```
codex --version
codex plugin marketplace add trailofbits/skills
codex plugin add static-analysis@trailofbits
codex plugin add audit-context-building@trailofbits
codex plugin add fp-check@trailofbits
codex plugin add variant-analysis@trailofbits
codex plugin add differential-review@trailofbits
codex plugin list --json
```

Für die zusätzlichen Prüfaufgaben werden auch `insecure-defaults@trailofbits` und `semgrep-rule-creator@trailofbits` mit `codex plugin add` eingerichtet. Die Handbook-Skills und die übrigen verlinkten Anleitungen werden über den im jeweiligen Host unterstützten Skillmechanismus geladen. Scannerprogramme, Sprachwerkzeuge, Build-Abhängigkeiten und Modellzugänge folgen separat. Ein Probelauf prüft, welche Programme starten, wo sie Dateien erzeugen und welche Rechte sie verwenden. Diese Installationsanleitung ist keine Behauptung, dass alle nativen Hostintegrationen im vorliegenden Versuch getestet wurden.

PRAXISANLEITUNG · INSTALLATION

Den Strix-Skill separat installieren

Für den einzelnen Strix-Skill nennt der Kickoff zusätzlich:

```
npx skills add usestrix/strix --skill penetration-testing-with-strix
```

Dieser Schritt installiert Skill-Inhalte. Strix, Docker und Modellkonfiguration folgen separat. Auch `npx` kann bei der Installation fremden Code ausführen und fällt unter die zuvor beschriebene Lieferkettenprüfung. Strix-Skills

Codex und Claude Code haben eigene Pluginmechanismen. Cursor unterstützt Agent Skills etwa unter `.cursor/skills/<name>/SKILL.md`. Eine portable Skilldatei garantiert keine gleiche Ausführung aller Zusatzaktionen, Teilagenten und Befehlsdateien auf jedem Host. Unter Windows benötigen Bash-basierte Bestandteile eine passende Umgebung; daraus folgt keine allgemeine Pflicht zur Windows-Linux-Umgebung WSL für jeden Scanner. Cursor Skills, Claude-Code-Plugins

Das Installationsprotokoll hält Host und Version, Herkunft und konkrete Paketrevision, Plugin-Versionen, verfügbare Programme, Lizenzen, Datenwege und den Probelauf fest. Verlinkte `main`-Seiten sind veränderlich. Die Testablage dokumentiert deshalb die tatsächlich verwendeten Quellen- und Werkzeugstände. Sie ersetzt nicht das Einrichtungsprotokoll Ihrer eigenen Agentenumgebung oder eine Prüfung der dort geladenen Plugins.

Für den hier dokumentierten Versuch wurden Anleitungen und Werkzeugstände am 29. Oktober 2026 gesondert festgehalten. Semgrep 1.182.1 lief in einer getrennten Ubuntu-/WSL-Umgebung, SkillSpector 2.12.0 in einer eigenen Python-Umgebung; die Codex-Security-CLI 0.8.1 brachte Plugin 0.1.95 mit. Hinzu kamen Clang 14, AFL++ 4.00c und CodeQL 2.27.3. Compiler und CodeQL wurden ausschließlich in die projektlokale Testumgebung ergänzt; Herkunft und Hashes sind protokolliert. Diese Stände und die beschriebenen Anpassungen gehören zu den Ergebnissen. Eine andere Installation kann andere Voraussetzungen und Ausgaben haben.

TECHNISCHE VERTIEFUNG

Prüfauftrag, Läufe und Befunde geordnet ablegen

PRAXISANLEITUNG · ABLAGEKONVENTION

Prüfauftrag, Läufe und Befunde geordnet ablegen

Außerhalb der geschützten Anwendungsarbeitskopie kann die folgende Struktur verwendet werden. Die Namen sind eine vorgeschlagene Konvention und keine automatisch erzeugten Standarddateien eines Produkts.

```
security-audit/<lauf-id>/
  auftrag.md            Revision, Umfang, Rechte, Eigentümer
  kontext.md            Zugänge, Datenflüsse, offene Annahmen
  laeufe/               Scannerkonfiguration, Status und Logs
  befunde/              Ein Fall je Kennung mit Belegverweisen
  tests/                Freigegebene Nachweise und Ergebnisse
  abdeckung.md          Untersuchte und ausgelassene Bereiche
  bericht.md            Bewertung und nächste Entscheidungen
```

Das führende Register kann im Ticketsystem bleiben und auf diese Artefakte verweisen. Die Plattform richtet den Zugriff ein; vertrauliche Belege behalten ihre Beschränkung auch beim Verlinken. AppSec hält den Prüfstatus aktuell, das Anwendungsteam übernimmt die Behebungsaufgaben.

FACHLICHE VERTIEFUNG

Pilotdaten für den Ausbau auswerten

Der Vergleich beginnt mit einer Scannerbasis auf derselben Revision. Danach wird die agentische Untersuchung ergänzt. Zur Entscheidung liegen Bericht, Befundregister, Abdeckungsübersicht und Aufwand vor. Erfasst werden zusätzliche bestätigte Fälle, widerlegte Meldungen, offene Kandidaten und nicht untersuchte Flächen sowie menschliche Prüf- und Behebungszeit, Scannerlizenzen, Modell- und Ausführungsressourcen. Grenzen und verfügbare Kapazität werden vor Beginn vereinbart; erst Einrichtung und Basismessung tragen einen belastbaren Terminplan.

Der Anteil bestätigter Fälle an fachlich abschließend bewerteten Meldungen beschreibt die Verwertbarkeit im Pilot. Er misst nicht alle übersehenen Fehler. Bekannte Schwachstellen oder ausdrücklich eingebrachte Testfälle helfen, Übersehenes sichtbar zu machen; ihre Erfolgsquote lässt sich nicht auf unbekannte Produktionsfehler übertragen. Ohne bekannten vollständigen Fehlerbestand ist eine Gesamterkennungsquote nicht seriös bestimmbar.

Nach Möglichkeit durchläuft mindestens ein ausreichend belegter Fall auch Behebung und Retest. Fehlt ein passender Befund, bleibt dieser Teil unbewertet oder wird mit einem ausgewiesenen Testfall erprobt. Ein Ausbau ist sinnvoll, wenn zusätzliche Erkenntnisse bestätigt werden, die Gegenprüfung tragbar bleibt und die Behandlung funktioniert. Sonst werden zuerst Prozesslücken geschlossen oder der Einsatz auf einen engeren Zweck begrenzt. Unbeantwortete Betriebsfragen, regelmäßig abgebrochene Läufe und fehlende Zuständigkeiten erhalten Aufgaben und Eigentümer.

Fazit

Am Dokumentenportal entscheidet sich der Nutzen an einer konkreten Frage: Kann das Team nachvollziehbar belegen, ob ein Kunde fremde Unterlagen abrufen kann, und eine erforderliche Korrektur überprüfen? Ein Agent kann den Weg durch den Code rekonstruieren, Analysewerkzeuge einsetzen und die Gegenprüfung vorbereiten. Eine belastbare Bewertung des Verdachts stützt sich auf Auftrag, Kontext und Belege; ob eine Korrektur wirkt, zeigt ein gesonderter Retest. Die Verantwortung für Zugriffsrechte, Freigaben und verbleibende Risiken liegt bei den benannten Menschen.

Die eigenen Versuche zeigen, weshalb diese Verbindung nötig ist. Bestehende Unit-Tests blieben trotz entfernter Eigentümerprüfungen grün. Bei der Untersuchung einer schädlichen Skillanleitung stand einem hohen Einzelbefund die Gesamtempfehlung SAFE gegenüber. Wer nur das grüne Testergebnis oder den zusammengefassten Score übernimmt, verliert den entscheidenden Zusammenhang. Ergebnisdateien müssen deshalb mit ihrem Laufstatus, ihren Einzelbelegen und den offenen Fragen gelesen werden.

Diese Beobachtungen stammen aus begrenzten Versuchen mit bekannten Fehlern und künstlichen Daten. Sie zeigen konkrete Ergebnisse und technische Lücken; eine Erkennungsleistung für unbekannte Produktionsfehler lässt sich daraus nicht ableiten. Nicht alle 18 Abläufe wurden vollständig getestet. Insbesondere die aktiven Strix- und Shannon-Läufe sowie ein abgeschlossener Codex-Security-Scan stehen weiterhin aus. Die dokumentierten Anleitungen und erwarteten Artefakte helfen bei der Vorbereitung, ersetzen aber den eigenen erfolgreich abgeschlossenen Lauf nicht.

Wenn Sie den Ansatz einführen, beginnen Sie mit einem Repository, einem bestätigten Bedrohungsmodell und einer klar begrenzten Sicherheitsfrage. Meine Empfehlung bleibt der statische Kern aus Trail of Bits, passenden CodeQL- oder Semgrep-Analysen und einer gesonderten Gegenprüfung mit `fp-check`. Verfolgen Sie mindestens einen geeigneten Fall bis zu Behebung und Retest. Wenn eine offene Frage einen aktiven Nachweis verlangt, können Sie Strix in einem gesondert freigegebenen Pilot erproben. Die Testumgebung und ihre technischen Grenzen müssen dafür bereitstehen.

Über den Ausbau entscheidet anschließend, ob zusätzliche bestätigte Erkenntnisse entstehen und das Team sie mit vertretbarem Aufwand bearbeiten kann. Der Maßstab ist ein Prüfweg, in dem aus einer plausiblen Behauptung eine nachvollziehbare Entscheidung und eine überprüfte Verbesserung wird. Daran sollte sich auch die nächste Erweiterung des Security-Stacks messen lassen.

Quellenstand und Aussagegrenzen

Grundlagenstand ist der 28. Oktober 2026; Testquellen und Werkzeugversionen wurden am 29. Oktober gesondert festgehalten. Der Portal-Fall ist synthetisch. Ausgeführt wurden unter anderem Semgrep und SkillSpector; der begonnene Codex-Security-Lauf scheiterte technisch. Die Ausführungsgrenzen und Ergebnisdateien stehen jeweils beim betreffenden Verfahren. Es wurden keine externen Anwendungen angegriffen oder produktiven Schwachstellen bestätigt. Ein Pilot muss die Wirksamkeit im eigenen Zielsystem prüfen.

Stilvorlage: „Enterprise Architecture für agentische KI“. Bildherkunft, Prompts und Diagrammquellen: `CREDITS.md`, `image-prompts.json`, `diagrams/`. Die generierten Szenen belegen keine Sicherheitswirkung.

Primärquellen

Die Quellen sind im Text an den jeweiligen Aussagen verlinkt. Dieses Verzeichnis enthält dieselben Nachweise zum Nachschlagen. Grundlagenprüfung: 28. Oktober 2026; ergänzende Testquellen und Ausführung: 29. Oktober 2026. Die konkreten Revisionen des Versuchs sind in dessen Quellenmanifest festgehalten.

1. Trail of Bits
   https://github.com/trailofbits/skills
2. Threat-Modeling-Skill
   https://github.com/openai/skills/tree/main/skills/.curated/security-threat-model
3. CodeQL
   https://docs.github.com/en/code-security/concepts/code-scanning/codeql/codeql-cli
4. Semgrep
   https://docs.semgrep.dev/semgrep-pro-vs-oss
5. statische Analyse
   https://github.com/trailofbits/skills/blob/main/plugins/static-analysis/README.md
6. Kontextaufbau
   https://github.com/trailofbits/skills/blob/main/plugins/audit-context-building/skills/audit-context-building/SKILL.md
7. Varianten
   https://github.com/trailofbits/skills/blob/main/plugins/variant-analysis/skills/variant-analysis/SKILL.md
8. Regelentwicklung
   https://github.com/trailofbits/skills/blob/main/plugins/semgrep-rule-creator/skills/semgrep-rule-creator/SKILL.md
9. unsichere Voreinstellungen
   https://github.com/trailofbits/skills/tree/main/plugins/insecure-defaults
10. Trail-of-Bits-Workflows
    https://github.com/trailofbits/skills/tree/2be6433b39e009cd2b6fd3be1895f6b1faa5af9a/plugins
11. Build-Modi
    https://docs.github.com/en/code-security/concepts/code-scanning/codeql/codeql-for-compiled-languages
12. Semgrep-Skill
    https://github.com/trailofbits/skills/blob/main/plugins/static-analysis/skills/semgrep/SKILL.md
13. CodeQL 2.27.3
    https://github.com/github/codeql-action/releases/tag/codeql-bundle-v2.27.3
14. öffentlichen MIT-Regressionsbeispiel
    https://github.com/github/codeql/tree/18b48aac513b7915adf742aa3fe226963a6ebcb6/python/ql/test/query-tests/Security/CWE-078-CommandInjection
15. Semgrep-Regelbedingungen
    https://semgrep.dev/legal/rules-license/
16. Plugin-Quickstart
    https://learn.chatgpt.com/docs/security/plugin
17. Cloud-Überblick
    https://help.openai.com/en/articles/20001107-codex-security
18. Deep-Scans
    https://learn.chatgpt.com/docs/security/plugin/deep-scans
19. CLI-Dokumentation
    https://learn.chatgpt.com/docs/security/cli

20. fp-check
https://github.com/trailofbits/skills/blob/main/plugins/fp-check/skills/fp-check/SKILL.md
21. Strix
https://github.com/usestrix/strix
22. Strix-Skill
https://github.com/usestrix/strix/blob/main/skills/penetration-testing-with-strix/SKILL.md
23. Shannon
https://github.com/KeygraphHQ/shannon
24. Konfiguration und Teilfehler
https://github.com/KeygraphHQ/shannon/blob/main/docs/configuration.md
25. Strix
https://github.com/usestrix/strix/blob/6b7ae2ebe4d47037a927aade23265ff4e4f27e84/skills/penetration-testing-with-strix/SKILL.md
26. Shannon
https://github.com/KeygraphHQ/shannon/blob/b9744f6cecc5429333873b86a6762f1d70c54fc2/docs/development.md
27. Testing-Handbook-Skills
https://github.com/trailofbits/skills/tree/main/plugins/testing-handbook-skills
28. libFuzzer
https://llvm.org/docs/LibFuzzer.html
29. SkillSpector
https://github.com/NVIDIA/skillspector
30. Skill Inspector
https://github.com/NVIDIA/SkillSpector/blob/b5987a332c9a118ec4714a5c79eb76641b546251/skills/skill-inspector/SKILL.md
31. MCP-Review
https://github.com/howardhsieh/agent-security-skills
32. Autorisierung
https://modelcontextprotocol.io/specification/2026-07-28/basic/authorization
33. Security Best Practices
https://modelcontextprotocol.io/docs/2026-07-28/tutorials/security/security_best_practices
34. SARIF 2.1.0
https://docs.oasis-open.org/sarif/sarif/v2.1.0/sarif-v2.1.0.html
35. Zugangsdaten
https://docs.github.com/en/code-security/concepts/secret-security/secret-scanning
36. Abhängigkeiten
https://docs.github.com/en/code-security/concepts/supply-chain-security/dependency-review
37. Änderungsprüfung
https://github.com/trailofbits/skills/blob/main/plugins/differential-review/skills/differential-review/SKILL.md
38. Code Scanning
https://docs.github.com/en/code-security/concepts/code-scanning/about-code-scanning
39. Strix-Skills
https://github.com/usestrix/strix/tree/main/skills
40. Cursor Skills
https://cursor.com/docs/skills
41. Claude-Code-Plugins
https://docs.anthropic.com/en/docs/claude-code/plugins