

Kann ein KI-Agent Software auf Sicherheitslücken untersuchen?

Cloudflare security-audit skill auf OWASP Juice Shop



confirmed

needs_validation

rejected

Das Ergebnis auf einen Blick

21

Verdachtsfälle, von zwei KI-Prüfern
bestätigt

0

bestätigte Schwachstellen: kein
Code wurde ausgeführt

320 / 320

zitierte Code-Stellen existieren und
stimmen



Die KI findet echte, gut nachvollziehbare Probleme.

Die Aufgabe des Seminars



Wie funktioniert eine KI-basierte Sicherheitsanalyse?



Welche Werkzeuge gibt es, und welche sind geeignet?



Wo liegen Stärken, Schwächen und Kosten?



Kann man die Funde verstehen und verifizieren?

Cloudflare security-audit skill, angewendet auf ein Open-Source-Projekt

Der security-audit skill: Anweisungen statt Programm



Was ist ein Skill?

Ein Ordner mit Anweisungen in Markdown. Der KI-Agent liest ihn und arbeitet Schritt für Schritt danach.

Kein Scanner, kein eigenes Programm. Von Cloudflare, MIT-Lizenz.

SKILL.md

Einstieg: Regeln, Profile, Budget, Schweregrade

RECONNAISSANCE, HUNTING, ATTACK-CLASSES

Aufklärung, Suche und Angriffsklassen

10 Begleitdateien

Spezialwissen, nur bei Bedarf geladen

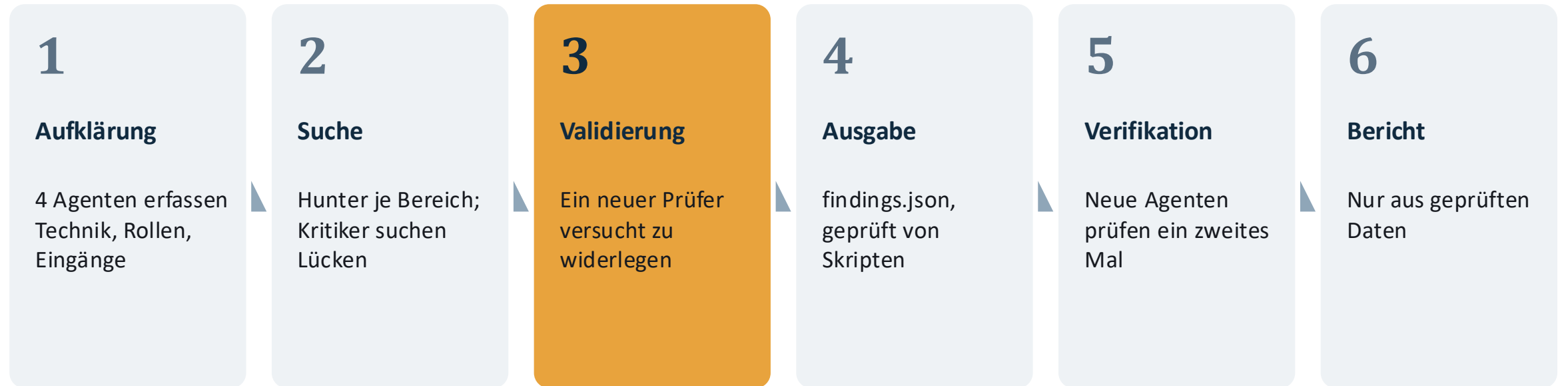
VALIDATION-AND-REPORTING

Prüfung, Verifikation und Report

Schema + 2 Validatoren

Prüfen die Ausgabe maschinell, ohne KI

Zwei Modi, Vier Rollen, sechs Phasen



Parent

koordiniert, schreibt als Einziger



Hunter

suchen Fehler



Prüfer

kennen die Suche nicht, widerlegen



Kritiker

suchen ungeprüften Code



Anleitungsmodus



Vollständiger Audit-Modus

Coverage Ledger, Profile und Budget



Coverage Ledger

Das Projekt wird in Prüfeinheiten zerlegt:

Eingang × Vertrauensgrenze × Teilsystem × Angriffsklasse

covered

geprüft, kein Fund

candidate

enthält einen Verdachtsfall

blocked

geprüft, Hinweise ungeprüft

deferred

zurückgestellt (Budget)

So sieht man auch, was nicht geprüft wurde.



Profile und Budget

quick

1 Suchwelle, 1 Kritiker, 1 Prüfer
pro Fall

Lauf 1

standard

Wellen bis die Kritiker nichts mehr
finden, 2 Prüfer pro Fall

Lauf 2

deep

feinere Einheiten, für große
Projekte

Budget = max. Zahl der Agentenaufrufe. Kritiker und Prüfer
werden zuerst reserviert.

Vier Regeln, die die KI ehrlich halten



Drei Urteile, Schweregrad nur mit Beweis

confirmed

Spur + lokal beobachtet

needs_validation

ein Fakt fehlt

rejected

von einem Prüfer widerlegt



Wer findet, bewertet nicht

Ein anderer Agent muss jeden Kandidaten zu widerlegen versuchen.



Abdeckung wird aufgeschrieben

Das Coverage Ledger zeigt, was geprüft, blockiert oder zurückgestellt wurde.



Ohne Sandbox keine Ausführung

Zielcode läuft nur isoliert: kein Netzwerk, nur lesen, harte Grenzen.

Das Zielprojekt: OWASP Juice Shop



Ein absichtlich unsicherer Webshop

- Gebaut für Sicherheitstrainings
- TypeScript, Express, Angular, SQLite
- Version 20.2.0, fester Commit 5658473c für alle Läufe
- „safetyMode“ schaltet gefährliche Challenges in Docker und Heroku ab
- Der letzte Commit sollte genau diesen Schutz verbessern

116

beabsichtigte Schwachstellen in
challenges.yml

Ein Lösungsschlüssel: Wir sehen, was die KI
findet und was sie übersieht.

Versuchsaufbau: zwei Läufe, eine Änderung

Lauf 1

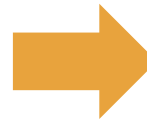
Profil quick

Budget **25 Agentenaufrufe**

Wellen **1 Suchwelle, 1 Kritiker**

Vorwissen **keins**

Ausführung **nur Quellcode**



Lauf 2

Profil standard

Budget **60 Agentenaufrufe**

Wellen **1 Suchwelle, 2 Kritiker**

Vorwissen **Lauf 1 (nur lesend)**

Ausführung **nur Quellcode**



Lauf 1: quick, Budget 25

26 min

Laufzeit

24 / 25

Agentenaufrufe

12

Verdachtsfälle



Budget-Entscheidung

Max. 2 Kandidaten pro Hunter. 23 weitere Ursachen blieben ungeprüft.



Prüfer korrigiert Hunter

Ein Windows-Pfadangriff wurde widerlegt und genauer formuliert.

Wer hat gearbeitet?

- 4 Agenten für die Aufklärung
- 6 Hunter in einer Suchwelle
- 1 Kritiker am Ende
- 13 Prüfer (12 + 1 erneute Prüfung)

Lauf 2: standard, Budget 60

3 h 13 min

Laufzeit

57 / 60

Aufrufe, + 20 Fehlstarts

21

Verdachtsfälle



Ratenlimit

Etwa 20 Agenten starteten gleichzeitig. Das Pro-Limit stoppte alle. Der Skill fragte nach. Neustart in Gruppen von 5.



Budget gebunden

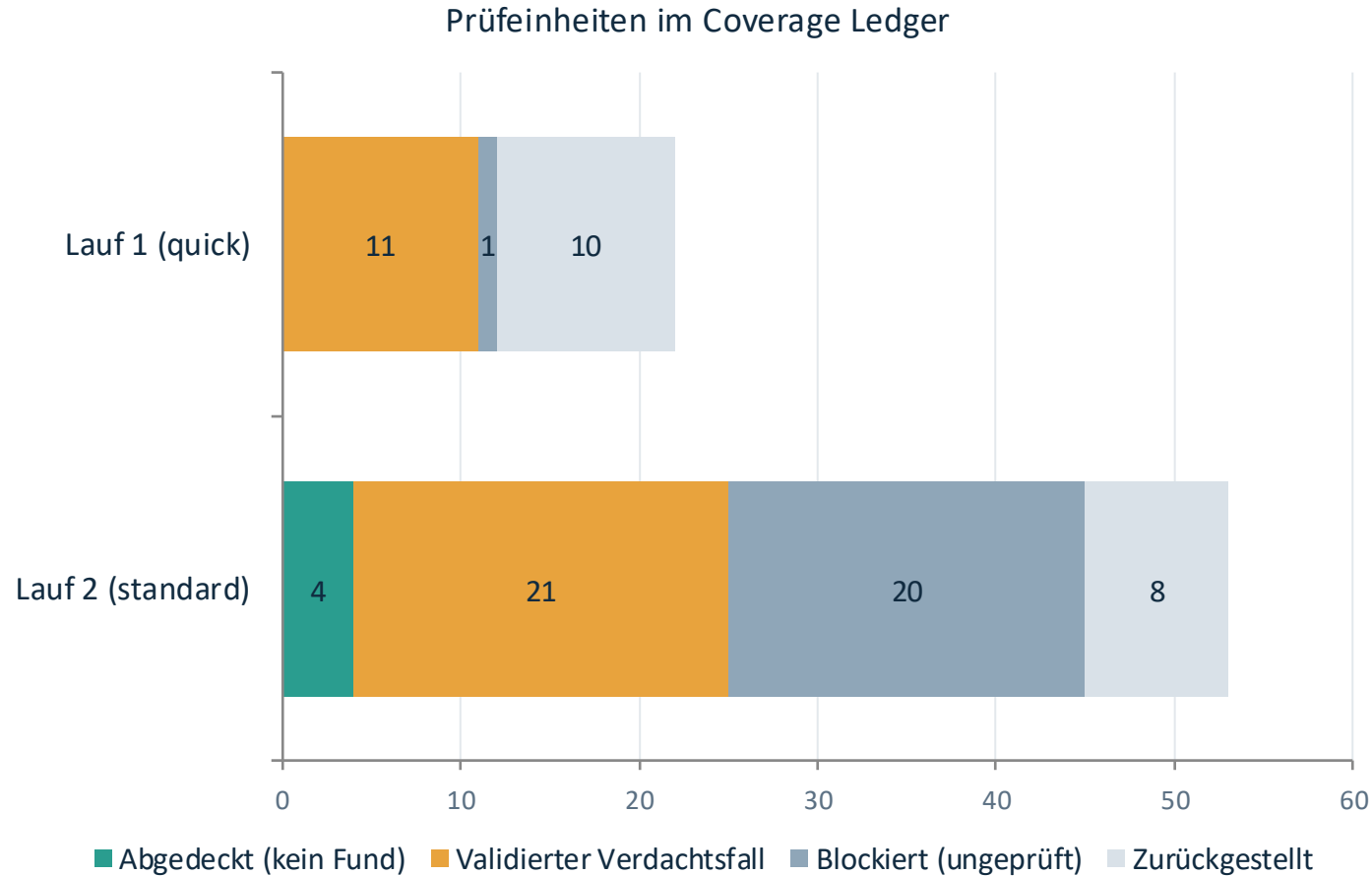
Die 12 alten Fälle brauchten 24 Prüfer. Darum nur 1 Kandidat pro Hunter; 33 Ursachen ungeprüft.



Sicherheitsfilter

Der erste Bericht wurde gestoppt. Er wurde ohne Angriffsdaten neu geschrieben.

Ergebnisse: mehr Breite, kein Beweis



12 → 21

validierte Verdachtsfälle

12 von 12

Fälle aus Lauf 1 kamen gleich zurück

5 + 4

neu: 5 aus Lauf 1 geprüft, 4 ganz neu

23 → 33

ungeprüfte Hinweise

Vier auffällige Verdachtsfälle



Sicherheitsschalter im neuesten Commit unvollständig

Der Schutz wirkt nur in einem Code-Zweig. Im anderen fehlen Sperrliste und Kürzung.

Nicht im Katalog

im Code bestätigt



YAML-Upload kann Code ausführen

js-yaml 3 baut aus YAML echte Funktionen. Der Upload braucht keine Anmeldung.

Nicht im Katalog

im Code bestätigt



Fotowand zeigt Passwort-Hashes

Ein öffentlicher Endpunkt liefert die vollen Benutzerdaten der Foto-Besitzer.

Nicht im Katalog

im Code bestätigt



Jeder kann Admin-Sitzungen erstellen

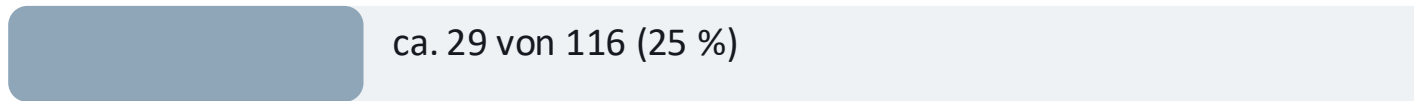
Der JWT-Schlüssel ist öffentlich; die Bibliothek nimmt auch Tokens ohne Signatur.

Im Katalog

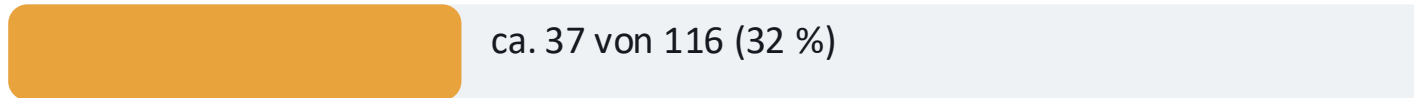
im Code bestätigt

Vergleich mit dem Lösungsschlüssel

Nach Lauf 1



Nach Lauf 1 und 2



Viele Challenges findet man gar nicht im Code: Foto-Ortung, Steganografie, schwache Passwörter. Zuordnung von mir, daher ungefähr.

12

Verdachtsfälle stehen nicht im Katalog

Einer davon war später falsch.

Verifikation in drei Ebenen

1

Format

Stimmt die Ausgabe mit dem Schema?

erledigt

Beide Prüfskripte bestanden. Sagt nichts darüber, ob ein Fehler echt ist.

2

Quellcode

Gibt es die Stellen, und tun sie das?

erledigt, nur lesend

320 von 320 Stellen stimmen. Bibliotheks-Code klärte 5 offene Fragen und widerlegte 1 Fall.

3

Laufzeit

Funktioniert der Angriff wirklich?

nicht gemacht

Braucht eine isolierte Umgebung. Mein Mac bietet sie nicht.

Ein Verdachtsfall ist kein Befund: das PATCH-Beispiel



Die Behauptung

PUT und DELETE sind gesperrt, aber vielleicht nicht PATCH. Dann könnte jeder Benutzerdaten ändern.



Zwei Prüfer behalten ihn

Beide fragen: „Kennt die Bibliothek PATCH?“ Die Bibliothek war nicht installiert.



Der Bibliotheks-Code widerlegt ihn

finale-rest 1.2.2 nutzt für Updates nur PUT. Eine PATCH-Route gibt es nicht.

Das Werkzeug war ehrlich: Es nannte genau, was es nicht prüfen konnte. Trotzdem war der Fall falsch. Die Genauigkeit muss man messen.

Stärken und Schwächen



Stärken

- Genau: 0 von 320 Code-Stellen erfunden
- Ehrlich: kein Schweregrad ohne Beweis
- Stabil: alle Fälle aus Lauf 1 kamen wieder
- Findet mehr als das Lehrbuch: 12 Fälle außerhalb des Katalogs
- Scheitert sicher: stoppt und fragt nach



Schwächen

- Ohne Sandbox keine Bestätigung
- Ohne Bibliotheks-Code bleiben Fragen offen
- Das Budget bestimmt, wie viel geprüft wird
- Viele parallele Agenten sprengen das Pro-Limit

Was es gekostet hat



51,65 \$

Lauf 2, geschätzt zu API-Listenpreisen (/usage)



3 h 13 min

Lauf 2, mit Wartezeit nach dem Ratenlimit



≈ 5,70 \$

pro neuem Verdachtsfall in Lauf 2

Fazit und nächste Schritte

Gut als Generator von Verdachtsfällen, noch keine Urteilsmaschine.

Der Skill findet echte, nachvollziehbare Probleme und sagt ehrlich, was er nicht beweisen kann. Für einen Beweis braucht es Ausführung oder einen Menschen.



**Ausführung in einer isolierten
Linux-VM**

Der Skill kann Fälle bestätigen oder
verwerfen. So misst man die Genauigkeit.



Ein zweites, kleines echtes Projekt

Gelten die Ergebnisse auch außerhalb
einer Trainings-App?

Vielen Dank!