Aktuelle Trends in der Zero-Day-Erkennung durch KI-Modelle

DFN-CERT Services GmbH

Version	Datum	Änderungen
1.0	15.5.2026	Initiale Version
1.1	22.5.2026	TLP:CLEAR-Einstufung hinzugefügt

Einleitung

Die Fähigkeiten der aktuellen Large-Language-Models (LLMs) werden immer besser, und die Effizienz und Breite der Einsatzmöglichkeiten wächst rapide. Ein aktueller Trend in der IT-Sicherheit ist der Einsatz dieser Modelle für die Erkennung von kritischen Schwachstellen im Quellcode von Programmen und Anwendungen. Laut Angaben des Herstellers Anthropic hat das neue Modell „Claude Mythos Preview“ bereits eine sehr große Anzahl an neuen Schwachstellen aufgedeckt und wird zunächst in der Glasswing Initiative nur einer begrenzten Anzahl von Akteuren im Bereich Betriebssysteme und Softwareentwicklung zur Verfügung gestellt.[1, 2] Die Vorstellung hierbei ist, dass die damit gefundenen Sicherheitsprobleme durch die Hersteller behoben werden können, bevor potenzielle Angreifer das Modell nutzen können.

In diesem Beitrag geben wir einen Überblick über die aktuellen Entwicklungen und schätzen ein, wie darauf zu reagieren ist. Wir beginnen mit einer Einführung der wichtigsten Begriffe und Grundlagen.

Zero-Day-Schwachstellen

Dies sind Schwachstellen in Software, die zum Zeitpunkt der Ausnutzung weder veröffentlicht noch einem größeren Kreis bekannt sind. Sie sind deshalb besonders kritisch, weil kein effektiver Schutz vor deren Ausnutzung verfügbar ist, zum Beispiel in Form eines Software-Patches.

Exploit-Code

Ein Programm oder Skript, das eine Schwachstelle ausnutzt, um Angreifern die Kontrolle über ein Zielsystem zu ermöglichen oder das Zielsystem zum Absturz zu bringen. Herausforderung bei der Entwicklung von Exploits sind die verschiedenen Sicherheitsmechanismen der verwendeten Betriebssysteme und Anwendungssoftware, die vor der Ausnutzung von Schwachstellen schützen sollen. Weiterhin werden in der Praxis in einem Angriff häufig mehrere Schwachstellen hintereinander ausgenutzt, um das Ziel — zum Beispiel Code-Ausführung — zu erreichen, was die Komplexität des Angriffs vergrößert

Suche nach Schwachstellen

Für die Suche nach Schwachstellen existieren mehrere Methoden. Eine davon ist die Untersuchung des Verhaltens der Software zur Laufzeit (dynamische Codeanalyse). Dabei wird versucht, durch entsprechend konstruierte Eingaben ein Verhalten zu provozieren, das auf das Vorhandensein einer Schwachstelle hinweist. Eine alternative Methode ist die Analyse des Quellcodes der Software. Da die Software dabei nicht ausgeführt wird, ist diese Methode als statische Codeanalyse bekannt. Im Folgenden geben wir einen schematischen Überblick über die Funktionsweise und Vor- und Nachteile dieser Methoden.

Fuzzing und dynamische Analyse

Durch Hersteller und Entwickler wird Software üblicherweise nur auf Eingaben getestet, die für deren Anwendung legitim sind. Dies betrifft sowohl die Länge als auch die Struktur der Eingabedaten. Idee des Fuzzings ist es, Eingaben zu produzieren und zu testen, die über diese zu erwartenden Eigenschaften hinaus gehen. Hierfür werden auf der Basis eines gezielten Prozesses zufällige Eingaben generiert, deren Struktur und Eigenschaften von den erwarteten Daten abweicht.

Bei Fehlern in der Speicherverwaltung durch die Software werden diese Daten nicht korrekt verarbeitet und führen in der Regel zum Absturz des Programms. Allerdings können diese Fehler sowie andere falsche Annahmen in der Software das Potential für Codeausführung haben, wenn ein funktionsfähiger Exploit generiert werden kann. Für eine weitere Analyse des Fehlers bieten sich Debugger an, über die die genauere Ursache des Absturzes und damit der Schwachstelle ermittelt werden kann. Die dynamische Analyse des Verhaltens ist auch eine effektive Methode, um Schwachstellen in Web-Anwendungen wie SQL-Injection und Cross-Site-Scripting zu finden.

Vorteil ist, dass sich diese Methode gut automatisieren lässt und die Ausnutzbarkeit der Schwachstellen direkt belegt. Allerdings gibt es auch einige Nachteile. So lassen sich prinzipbedingt kaum Schwachstellen in der Programmlogik finden. Ein Beispiel hierfür sind Fehler bei der Authentifizierung und Zugriffskontrolle. Des Weiteren ist eine Erkennung von Schwachstellen nicht zuverlässig möglich, die über mehrere, logisch aufeinanderfolgende Eingabedaten ausgenutzt werden müssen.

Statische Code-Analyse

Bei der statischen Code-Analyse wird nach Mustern im Quell-Code gesucht, die typisch für Schwachstellen sind. Der einfachste Ansatz ist die Suche nach bekannten, feststehenden Mustern wie Default-Passwörter, die fest im Code integriert sind. Ein besserer Ansatz ist es, die Abhängigkeiten im Code inklusive des Datenflusses innerhalb und zwischen Funktionen zu berücksichtigen. Damit lassen sich zum Beispiel Schwachstellen erkennen, die durch eine unzureichende Prüfung von Größe und Inhalt der Eingabedaten verursacht werden und die von einem Angreifer manipuliert werden können (zum Beispiel Buffer-Overflow-Schwachstellen).

Vorteil ist die direkte Erkennung von typischen Fehlern im Programmcode. Dadurch lassen sich in einem bestimmten Umfang auch logische Fehler aufdecken, die zum Beispiel durch die unzureichende Authentifizierung oder Autorisierung von Benutzern verursacht werden. Des Weiteren lassen sich damit grundlegende Fehler bei der Speicherverwaltung erkennen. Komplexe Fehler, deren Ausnutzung mehrere aufeinanderfolgende Schritte benötigt, lassen sich aber durch Muster-basierte Erkennung nur bedingt finden.

Da Large-Language-Models auf umfangreiche Code-Fragmente und Muster für Schwachstellen trainiert werden, können diese zur Erkennung von Fehlern verwendet werden. Aufgrund der Eigenschaften der LLMs, Schlussfolgerungen ziehen zu können (Reasoning), lassen sich dadurch auch vielschichtigere Sicherheitsprobleme aufdecken. Allerdings kann eine rein statische Analyse unter Verwendung von Large-Language-Models deren Ausnutzbarkeit im Allgemeinen nicht direkt belegen. Insbesondere neigen LLMs zu einer falschen Diagnose von Softwarefehlern, wodurch korrekter Code als Schwachstelle klassifiziert wird (False Positive).

Aktuelle Verbesserungen bei der Anwendung von LLMs zur Suche nach Schwachstellen

Large-Language-Models (LLMs) sind KI-Modelle aus dem Bereich des NLP (Natural Language Processing) und erzeugen Antworten auf Prompts auf der Basis eines komplexen statistischen Ansatzes über Text und Programmcode. Aus deren Funktionsweise ergibt sich die inhärente Neigung zu Halluzinationen, also unzutreffenden Aussagen. LLMs bieten zwar eine sehr gute Mustererkennung für verdächtige Code-Segmente, allerdings hat die Neigung zu Halluzinationen in der Vergangenheit zu einer sehr hohen Meldung an fehlerhaften Schwachstellen-Meldungen geführt.[3]

Dies hat sich mit der Einführung der aktuellen Generation von KI-Modellen und deren Einsatz im Rahmen von Agentensystem deutlich geändert. Die Verbesserungen gipfeln aktuell in „Claude Mythos Preview“ von Anthropic, das im Moment aufgrund seiner Kritikalität nur ausgewählten Anwendern zur Verfügung gestellt wird.[1, 2] Allerdings sind auch andere Modelle in der Lage, Zero-Day-Schwachstellen zu finden.[4]

Die signifikanten Verbesserungen von Claude Mythos Preview und anderen KI-Modellen der bekannten Anbieter sind durch zwei Entwicklungen erreicht worden:

- **Fine-Tuning der Modelle:** Die Modelle werden durch Fine-Tuning für Software, bekannte Schwachstellen und die Entwicklung von Exploit-Code weiter optimiert. Dies resultiert in einer verbesserten Erkennung potentieller Softwarefehler. Weiterhin unterstützt dies die autonome Entwicklung von Exploits zum Ausnutzen von Schwachstellen.
- **Einsatz von agentischen Architekturen:** Ein allgemeiner Trend bei der Verwendung von LLMs ist der Einsatz von einem oder mehreren KI-Agenten (Agentic AI). Ein Agent ist eine Software-Komponente, die
  - Zugriff auf verschiedene Datenquellen hat,
  - Software-Werkzeuge für spezielle Anwendungen nutzen kann,
  - ein LLM zur Planung der Durchführung von Aufgaben nutzt und
  - ein LLM für die Interpretierung von textuellen Ein- und Ausgaben verwendet.

  Das LLM übernimmt damit die Rolle der kognitiven Komponente, die Aufgaben planen, Daten und Tool-Ausgaben interpretieren und eine textuelle Antwort der erzielten Ergebnisse erzeugen kann. Der Agent dagegen ist die Schaltzentrale, die Zugriff auf die für die Aufgabe benötigten Daten hat, Software-Tools gezielt aufrufen und die vom LLM vorgeschlagenen Entscheidungen umsetzen kann.

Wie die Übersicht der Methoden zur Entdeckung von Schwachstellen zeigt, können LLMs zwar sehr gut Muster für komplexere Schwachstellen entdecken, es besteht aber eine größere Wahrscheinlichkeit für False Positives. Zwar ermöglicht die dynamische Analyse den direkten Nachweis einer Schwachstelle, allerdings fehlen die Mittel, komplexere Fehler zu finden. Mittels des Einsatzes von KI-Agenten lassen sich die Vorteile der statischen und dynamischen Analyse in einem mehrstufigen Prozess vereinen:

- Ein Agent nutzt das LLM, um die Struktur des untersuchten Software-Projekts zu verstehen und darauf aufbauend Kandidaten für Schwachstellen zu bestimmen.
- Im nächsten Schritt nutzt der Agent das LLM, um Schritte zum Ausnutzen der Schwachstelle — also entsprechend konstruierte Eingabedaten oder einen Exploit — zu generieren. Die Idee ist vergleichbar mit dem Fuzzing, nur dass hierbei Exploit-Daten zielgerichtet auf der Basis der Ergebnisse der vorherigen Code-Analyse erstellt werden, die zum Beispiel den Fluss kritischer Daten im Programm beinhalten.
- Der Agent startet und überwacht das Verhalten des Programms mittels eines Debuggers. Das LLM nutzt die Ausgabe des Debuggers, um die Ausnutzbarkeit der Schwachstelle zu verifizieren und optional angepassten Exploit-Code zu generieren. Dieser Ansatz wurde zum Beispiel von Anthropic bei der Anwendung von Claude Mythos Preview eingesetzt.[2]

Die prinzipielle Schwäche von LLMs, die zu Halluzinationen und fehlerhaften Voraussagen führt, wird also durch zusätzliche, aus dem Software-Engineering bereits bekannte Werkzeuge mitigiert. Durch den Prozess können auch komplexere Fehler gefunden werden, und die Generierung von mehrstufigen Exploits wird ermöglicht. Ein Beispiel ist in dem Blog-Artikel von Anthropic beschrieben worden.[5]

Fazit und Empfehlungen

In der Vergangenheit sind kritische Schwachstellen regelmäßig entdeckt und ausgenutzt worden. Das betrifft sowohl Server- als auch Client-Anwendungen wie beispielsweise Webserver und Internetbrowser.[6] Zumindest eine Teilmenge davon ist im Rahmen von Angriffen ausgenutzt worden, bevor diese Schwachstellen durch den Hersteller, CERTs oder Dritte veröffentlicht wurden (Zero-Days). Ein effektives Schwachstellen- und Patchmanament sowie eine Reaktion auf Bedrohungen durch

Zero-Day-Schwachstellen ist also schon vor der Einführung der aktuellen KI-Modelle zur Erkennung von Sicherheitsproblemen eine bedeutende Herausforderung in der IT-Sicherheit gewesen. Weiterhin ist es unwahrscheinlich, dass neuartige Klassen von Schwachstellen oder andere kritische Sicherheitsprobleme exklusiv durch KI-Modelle gefunden werden.[12]

Allerdings sind die beschriebenen Fähigkeiten der aktuellen KI-Modelle wie Claude Mythos Preview und agentischer Architekturen plausibel und es ist damit zu rechnen, dass die Rate der gefundenen und gemeldeten Schwachstellen (inklusive Zero-Day-Schwachstellen) sehr stark ansteigt. Weiterhin ist anzunehmen, dass sich die Modelle und Ansätze kontinuierlich verbessern werden. Dies ist im Wesentlichen darin begründet, dass die aktuellen LLMs der führenden Hersteller immer gezielter auf Code und Schwachstellen optimiert werden und der agentische Ansatz zur Vermeidung von False Positives und Erzeugung von Exploit-Code weiter verbessert wird.

Wie Beiträge inklusive [7] zeigen, ist die automatische Erstellung von Exploits bereits ohne den Einsatz von LLMs umgesetzt worden. Allerdings ist auch hier eine deutliche Verbesserung durch die generativen KI-Modelle zu erwarten, deren Fähigkeiten die Erzeugung von komplexen, mehrstufigen Exploits ermöglichen. Dafür werden mehrere Schwachstellen zum Erreichen eines Ziels kombiniert und die verwendeten Sicherheitsmechanismen können umgangen werden. Es ist also damit zu rechnen, dass komplexe Exploits effektiv generiert werden und die Zeitspanne zwischen dem Auffinden und Ausnutzen einer Schwachstelle weiter abnimmt oder sogar durch autonome Exploit-Erzeugung ganz verschwindet.[8] Dies ist insbesondere deshalb kritisch, weil bestehende KI-Modelle die weitgehende Automatisierung von mehrstufigen Angriffen ermöglichen, die die Kill Chain vollständig abbilden, und damit beispielsweise Ransomware-Angriffe wesentlich erleichtern.

Die Kombination aus der Vielzahl neu entdeckter Schwachstellen, deren autonomer Ausnutzung und der Automatisierung der Angriffskette führt dazu, dass eine direkte manuelle Reaktion auf neue Schwachstellen immer schwieriger wird. Eine spezifische Reaktion auf diese Tendenzen ist sicherlich — wenn überhaupt möglich — sehr schwierig. Trotzdem ist eine Reihe von allgemeinen Maßnahmen zu empfehlen, mit denen die eigene Sicherheit proaktiv wesentlich erhöht werden kann:

- Prüfen, ob Prozesse für das Schwachstellen- und Pachtmanagement vorhanden sind, und diese gegebenenfalls weiter optimieren. Ziel ist es, möglichst schnell Bedrohungen durch neue Schwachstellen erkennen und darauf reagieren zu können. Dafür lassen sich potentiell auch KI-Modelle einsetzen, die Prozesse automatisieren oder zumindest unterstützen.
- Netzwerke und Systeme härten und Defense-in-Depth einführen, um Angreifern das Erreichen ihrer Zielsetzungen nach einem erfolgreichen initialen Angriff zu erschweren.

- Systeme zur Erkennung von Angriffen aktuell halten und die aktuellen Tendenzen im Blick behalten (Threat Intelligence); zum Beispiel bietet das DFN-CERT regelmäßig Lageberichte an.[6]
- Sich auf zukünftige Krisen vorbereiten. Der zu erwartende deutliche Anstieg an Zero-Day-Schwachstellen und die damit korrespondierenden Exploits machen komplexe kritische Vorfälle deutlich wahrscheinlicher. Die Vorbereitung auf derartig hervorgerufene Krisen wird in Zukunft immer wichtiger.[9]

Eine positive Tendenz ist aber, dass diese KI-Modelle auf längere Sicht die Sicherheit der Software erhöhen können. Neben der autonomen Erstellung von Exploits sind die Modelle auch in der Lage, effektive Patches zum Schließen von Schwachstellen zu generieren. Dies ist insbesondere dann hilfreich, wenn die Hersteller und Entwickler die Modelle frühzeitig nutzen, um Patches für gefundenen Schwachstellen zur Verfügung zu stellen. Die verbliebenen Sicherheitsprobleme lassen sich danach nur mit einem sehr hohen Aufwand erkennen, den Angreifer häufig nur schwer leisten können.

Eine ausführliche Zusammenfassung der aktuellen Situation und Reaktion ist in einem Report der Cloud Security Alliance und einem SANS-Webinar veröffentlicht worden.[10, 11]

Referenzen

[1] Projekt Glasswing von Anthropic: https://www.anthropic.com/glasswing

[2] Assessing Claude Mythos Preview's cybersecurity capabilities, https://red.anthropic.com/2026/mythos-preview/

[3] Xueying Du, Jiayi Feng, Yi Zou, Wei Xu, Jie Ma, Wei Zhang, Sisi Liu, Xin Peng, Yiling Lou, Reducing False Positives in Static Bug Detection with LLMs: An Empirical Study in Industry, Januar 2026, https://arxiv.org/abs/2601.18844

[4] Evaluating and mitigating the growing risk of LLM-discovered 0-days, https://red.anthropic.com/2026/zero-days/

[5] Reverse engineering Claude's CVE-2026-2796 exploit, https://red.anthropic.com/2026/exploit/

[6] Christine Kahl, Neues aus dem DFN-CERT, https://www.dfn.de/wp-content/uploads/2026/01/bt84_forum_sicherheit_dfn_cert-VORTRAG.pdf

[7] Hong Hu, Zheng Leong Chua, Sendroiu Adrian, Prateek Saxena, and Zhenkai Liang, Automatic Generation of Data-Oriented Exploits, Usenix Security Conference 2015, https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-hu.pdf

[8] From Vulnerability to Exploitation, https://zerodayclock.com/

[9] TALON 2026: European Crisis Simulation for NRENs, https://security.geant.org/talon-2026/

[10] The "AI Vulnerability Storm": Building a "Mythos-ready" Security Program, https://labs.cloudsecurityalliance.org/mythos-ciso/

[11] SANS-Warnhinweis: BugBusters – KI-Schwachstellenerkennung: Hype versus Realität, https://www.youtube.com/watch?v=X0aik3eCTdU

[12] Bobby Holley, The zero-days are numbered, https://blog.mozilla.org/en/firefox/ai-security-zero-day-vulnerabilities/