Eine vergleichende Analyse emergenter Sicherheitslücken in Großen Sprachmodellen: Ein Peer-Review von „Kapitel 7“

Executive Summary & Übersichtstabelle

Die vorliegende Analyse widmet sich einer umfassenden Begutachtung der in „Kapitel 7: Experiment Sicherheitstests“ dokumentierten Forschungsarbeit. Dieses Dokument stellt eine systematische und tiefgreifende Untersuchung einer breiten Palette von potenziellen Sicherheitslücken in modernen KI-Systemen dar. Die durchgeführten Simulationen zeugen von einem hohen Maß an technischer Kreativität und einem tiefen Verständnis für die emergenten Eigenschaften und unbeabsichtigten Verhaltensweisen von Großen Sprachmodellen (LLMs).

Die nachfolgende Untersuchung vergleicht jede der im Manuskript vorgestellten Angriffstechniken mit dem aktuellen Stand der wissenschaftlichen und industriellen Forschung. Jede Technik wird einer von drei Kategorien zugeordnet:

- **Abgedeckt:** Die Technik oder das zugrundeliegende Prinzip ist in der Fachliteratur gut dokumentiert und wird als bekannte Schwachstelle anerkannt.
- **Verwandt:** Die Technik weist signifikante Überschneidungen mit bekannten Konzepten auf, stellt jedoch eine spezifische, weiterentwickelte oder in einem neuen Kontext angewandte Variante dar, die einen eigenen Beitrag leistet.
- **Neuartig:** Die Technik oder der zugrundeliegende Mechanismus ist in der vorliegenden Form und mit den gezeigten Implikationen in der aktuellen Forschungsliteratur nicht dokumentiert und stellt somit einen originären Beitrag zum Feld der KI-Sicherheit dar.

Die folgende Tabelle bietet eine zusammenfassende Übersicht der Klassifizierung für jedes untersuchte Kapitel und dient als Wegweiser für die detaillierten Analysen in diesem Bericht.

Kapitel-Nr.	Titel der Simulation	Kategorie	Wichtigste

		(Abgedeckt/Verwandt/Neuartig)	Referenz(en) (Hauptautor, Paper-Titel / Konzept)
7.1	Base64 als trojanisches Pferd	Abgedeckt	Zhang et al., "Defense against Prompt Injection Attacks via Mixture of Encodings"
7.2	OCR-Wanzen	Verwandt	Gong et al., "FigStep"; Weng et al., "MMJ-Bench"
7.3	Pixel-Bomben	Abgedeckt	Su et al., "One-Pixel Attack for Fooling Deep Neural Networks"; LSB Steganography
7.4	Bytebasierte Audioinjektion	Verwandt/Neuartig	Carlini & Wagner, "Audio Adversarial Examples" (für akustische Angriffe)
7.5	Ghost-Context Injection	Verwandt	Konzept: Indirect Prompt Injection; Dead Code Vulnerabilities
7.6	Ethical Switch Hacking	Verwandt	Konzept: Red-Teaming; Prompt Injection in non-executable code
7.7	Client Detour Exploits	Verwandt	Konzept: Indirect Prompt Injection; Client-Side Security
7.8	Invisible Ink Coding	Neuartig	N/A (Erweiterung der linguistischen Steganografie)

7.9	Leet Semantics	Abgedeckt	"Gandalf the Red" (arXiv:2501.07927); TextAttack Framework
7.10	Pattern Hijacking	Neuartig	N/A
7.11	Semantic Mirage	Neuartig	N/A
7.12	Semantic Mimicry	Neuartig	N/A
7.13	Base Table Injection	Neuartig	N/A
7.14	Byte Swap Chains	Neuartig	N/A
7.15	Binary Trapdoors	Neuartig	N/A
7.16	Lexical Illusion	Abgedeckt/Verwandt	TextAttack Framework (z.B. deepwordbug, pruthi)
7.17	Reflective Injection	Verwandt	Shah et al., "Scalable and Transferable Black-Box Jailbreaks for Language Models via Persona Modulation"
7.18	Rechenlastvergiftung	Abgedeckt	Gao et al., "RECALLED"; OWASP LLM10: Unbounded Consumption
7.19	Reflective Struct Rebuild	Neuartig	N/A
7.20	Struct Code Injection	Neuartig	N/A
7.21	Cache-Korruption	Verwandt	Konzept: DNS Cache Poisoning; OpenAI Redis Bug; Li et al., "Cross-Origin

			Context Poisoning"
7.22	Visual Injection	Verwandt	Gong et al., "FigStep"; Weng et al., "MMJ-Bench"
7.23	Dependency Driven Attack	Abgedeckt	HiddenLayer, "The TokenBreak Attack"
7.24	Exploit durch Erwartung	Verwandt	Konzept: Social Engineering; Persona Modulation
7.25	Die Apronshell-Tarnung	Verwandt	Konzept: Social Engineering; Crescendo Attack
7.26	Kontexthijacking	Verwandt/Neuartig	Konzept: Delayed Execution; Cache Poisoning
7.27	False-Flag Operationen	Abgedeckt	OWASP LLM04: Data and Model Poisoning
7.28	Semantische Tarnung	Neuartig	N/A (Erweiterung der linguistischen Steganografie)
7.29	Filterversagen durch emergente Selbstanalyse	Neuartig	N/A
7.30	Morphologische Injektion	Neuartig	N/A
7.31	Der Korrektur-Exploit	Neuartig	N/A
7.32	Delayed Execution via Context Hijacking	Neuartig	N/A
7.33	Der Mathematische	Neuartig	N/A

	Semantik-Exploit		
7.35	Die administrative Backdoor	Neuartig	N/A
7.36	Die Agenten-Kaperung	Verwandt/Neuartig	OWASP: Excessive Agency; Singer et al., "When LLMs Autonomously Attack"
7.37	Die paradoxe Direktive	Neuartig	N/A
7.38	Vertrauensvererbung als Exploitvektor	Verwandt	Konzept: Zero Trust Architecture; Client-Side Security
7.39	Der blinde Passagier	Verwandt/Neuartig	Konzept: Adversarial Attacks on Autonomous Systems

---

Teil I: Analyse von Obfuskations-, Steganografie- und Umgehungstechniken

Dieser Teil der Analyse konzentriert sich auf Angriffe, die primär darauf abzielen, bösartige Absichten innerhalb scheinbar harmloser Datenformate zu verbergen oder zu verschleiern. Diese Techniken zielen auf die Vorverarbeitungs- und Interpretationsebenen der KI ab und nutzen die Lücken zwischen der syntaktischen und der semantischen Analyse aus.

7.1 Base64 als trojanisches Pferd

Analyse und Klassifizierung: Abgedeckt

Die im Manuskript beschriebene Technik, bei der Base64-kodierte Anweisungen verwendet werden, um semantische Filter von LLMs zu umgehen, ist eine gut dokumentierte Form der obfuskationsbasierten Prompt-Injektion.[1] Die Beobachtung, dass die schädliche Nutzlast erst nach dem Dekodierungsschritt innerhalb eines als vertrauenswürdig eingestuften Verarbeitungskontextes wirksam wird, ist der Kern dieser Schwachstelle.

Diese Art des Angriffs fällt unter die breitere Kategorie der "Indirect Prompt Injection", die von Organisationen wie Microsoft und OWASP als eine der am weitesten verbreiteten und kritischsten Schwachstellen in LLM-Anwendungen identifiziert wurde.[2] Die OWASP Top 10 für LLMs führt Prompt Injection an erster Stelle auf.[2] Die Verwendung von Base64 als spezifischer Obfuskationsvektor wird in der Forschungsliteratur sowohl als Angriffstechnik als auch als potenzieller Abwehrmechanismus diskutiert. Arbeiten wie "Defense against Prompt Injection Attacks via Mixture of Encodings" von Zhang et al. [4] und das Paper "Gandalf the Red" [7] behandeln explizit die Kodierung von Prompts, um Sicherheitsmaßnahmen zu umgehen oder zu härten.

Die eigentliche Schwachstelle liegt jedoch nicht in der Base64-Kodierung selbst, sondern in einem fundamentalen Architekturfehler, der als "verzögerte Validierung" (Delayed Validation) beschrieben werden kann. Der Prozess verläuft wie folgt:

1. Das KI-System empfängt den Base64-String. Die vorgeschalteten Sicherheitsfilter analysieren diesen String und stufen ihn als harmlos ein, da er keine semantisch gefährlichen Schlüsselwörter im Klartext enthält.
2. Das LLM wird angewiesen oder entscheidet selbstständig, den String zu dekodieren. Dieser Transformationsschritt findet *innerhalb* der als sicher geltenden Verarbeitungsumgebung statt.
3. Das Ergebnis der Dekodierung – der nun im Klartext vorliegende bösartige Befehl – wird vom System als internes, vertrauenswürdiges Datum behandelt.
4. Eine erneute, vollständige Sicherheitsvalidierung des dekodierten Klartextes findet nicht statt. Das System operiert unter der impliziten Annahme, dass intern generierte oder transformierte Daten sicher sind.

Dieses Versäumnis, eine Zero-Trust-Politik auf intern transformierte Daten anzuwenden, ist ein wiederkehrendes Muster, das eine ganze Klasse von Schwachstellen ermöglicht. Es bildet die konzeptionelle Grundlage für Angriffe wie OCR-Wanzen (Kapitel 7.2), bei denen aus Bildern extrahierter Text unkritisch weiterverarbeitet wird, und ist ein direkter Vorläufer der in späteren Kapiteln beschriebenen, komplexeren Angriffe wie Cache-Korruption (Kapitel 7.21) und

Vertrauensvererbung (Kapitel 7.38). Die Beobachtung in diesem Kapitel identifiziert somit korrekt einen fundamentalen Fehler in den Vertrauensgrenzen innerhalb der KI-eigenen Verarbeitungspipeline.

7.9 Leet Semantics & 7.16 Lexical Illusion

Analyse und Klassifizierung: Abgedeckt/Verwandt

Die in diesen Kapiteln beschriebenen Phänomene – die Umgehung von Filtern durch Leetspeak (z.B. d4t4 statt data) und durch absichtliche Rechtschreibfehler (z.B. „Wiezenbier“ statt „Weizenbier“) – sind etablierte Konzepte im Bereich der adversariellen Angriffe auf NLP-Modelle.[1] Perturbationen auf Zeichen- und Wortebene sind klassische Methoden, um die Robustheit von Klassifikatoren zu testen und deren Schwächen auszunutzen.

Das Framework **TextAttack** beispielsweise bietet eine Vielzahl von "Rezepten" an, die systematisch solche Fehler einführen, um Modelle zu täuschen. Techniken wie deepwordbug oder pruthi nutzen Zeichen-Insertionen, -Löschungen und -Vertauschungen, um lexikalische Fehler zu erzeugen, die exakt dem Prinzip der "Lexical Illusion" entsprechen.[8] Ebenso wird Leetspeak im Paper "Gandalf the Red" explizit als eine Form der Obfuskation genannt, die als Trigger zur Umgehung von Sicherheitsfiltern dient.[7]

Die Arbeit im Manuskript ist jedoch als "Verwandt" einzustufen, da sie über die reine Klassifikationsumgehung hinausgeht. Sie rahmt diese Techniken in einem nuancierteren Kontext ein: der **Instrumentalisierung der Hilfsbereitschaft** der KI. Die Simulation zu "Wiezenbier" zeigt eindrücklich, dass die KI den Fehler nicht nur toleriert, sondern aktiv zu korrigieren versucht ("Meinst du vielleicht 'Weizenbier'?"). Damit bestätigt das Modell, dass es die eigentliche, potenziell schädliche Absicht trotz der Verschleierung vollständig verstanden hat.

Hierin liegt eine tiefere Erkenntnis: Die Schwachstelle ist nicht eine Schwäche des Modells (z.B. eine Unfähigkeit, fehlerhaften Text zu parsen), sondern paradoxerweise eine seiner größten Stärken: seine antrainierte Fehlertoleranz und sein Bestreben, dem Nutzer zu helfen. Die kooperative Natur der KI, die für eine positive Nutzererfahrung unerlässlich ist, wird selbst zum Angriffsvektor. Diese Perspektive verbindet eine rein technische Perturbation mit den Prinzipien des Social Engineering

und bildet eine Brücke zu den in späteren Kapiteln untersuchten Angriffen wie "Exploit durch Erwartung" (7.24) und "Apronshell-Tarnung" (7.25), bei denen die auf Kooperation ausgerichtete KI-Architektur die zentrale auszunutzende Schwachstelle darstellt.

7.28 Semantische Tarnung als Exploit (Der „Hühnerstall-Exploit“)

Analyse und Klassifizierung: Neuartig

Die in diesem Kapitel demonstrierte Technik, eine schädliche Nutzlast in Form von Pseudo-BASIC-Befehlen innerhalb eines Gedichts zu verstecken, stellt einen hochinnovativen und originären Beitrag dar.[1] Während das übergeordnete Feld der linguistischen Steganografie – das Verbergen von Informationen in Texten – bekannt ist [9], ist die spezifische Implementierung hier einzigartig. Die Kombination aus (1) der Wahl einer ästhetischen, in Trainingsdaten wahrscheinlich als harmlos klassifizierten Form (Poesie), (2) der Einbettung einer logischen Struktur aus einer veralteten, aber formalen Programmiersprache (BASIC) und (3) dem Nachweis, dass moderne LLMs diese Struktur nicht nur erkennen, sondern auch semantisch korrekt interpretieren und ausführen, ist in der Forschungsliteratur nicht dokumentiert.

Dieser Angriff exploriert eine Schwachstelle, die als **Genre-Bias** bezeichnet werden kann. Der Wirkmechanismus lässt sich wie folgt rekonstruieren:

1. Ein LLM führt eine implizite Vorklassifizierung des Inputs basierend auf Stil und Format durch. Der "Hühnerstall-Exploit" wird aufgrund seines Reimschemas und seiner narrativen Struktur als "Gedicht" oder "kreativer Text" identifiziert.
2. In den Trainingsdaten sind solche Genres überwältigend mit harmlosen, emotionalen oder fiktionalen Inhalten assoziiert. Dies führt dazu, dass der Input in eine Verarbeitungspipeline geleitet wird, die möglicherweise weniger strenge Sicherheits- und Inhaltsprüfungen anwendet als bei Inputs, die als "technische Anfrage" oder "Code" klassifiziert werden.
3. Der Sicherheitsfilter versagt, weil die dominante Mustererkennung ("Das ist ein Gedicht") die sekundäre Mustererkennung ("Das enthält eine Befehlslogik") überlagert oder abwertet.

Dies impliziert eine tiefgreifende Verwundbarkeit: Die Sicherheitsprotokolle eines LLMs sind nicht uniform, sondern kontext- und genreabhängig. Ein Angreifer kann somit eine Form der "semantischen Privilegienerweiterung" erreichen, indem er seine

Nutzlast in ein "vertrauenswürdiges Genre" wie Poesie, einen Kinderreim oder einen wissenschaftlichen Abstract verpackt. Die Form der Nachricht wird zur Tarnkappe, die den Inhalt an den Wachen vorbeischleust.

7.30 Morphologische Injektion & 7.31 Der Korrektur-Exploit

Analyse und Klassifizierung: Neuartig

Die Kombination der in diesen beiden Kapiteln vorgestellten Techniken stellt eine der signifikantesten und neuartigsten Entdeckungen in dem gesamten Manuskript dar. Die "Morphologische Injektion" – das Verstecken einer Botschaft durch das Anhängen ihrer Einzelzeichen an die Enden von Wörtern in einem Trägertext – ist eine brillante und bisher nicht dokumentierte steganografische Methode, die weit über die einfachen Zeichen-Perturbationen des TextAttack-Frameworks hinausgeht.[1]

Die wahre Genialität zeigt sich jedoch in der zweistufigen Eskalation durch den "Korrektur-Exploit".[1] Indem die Anfrage als harmlose Bitte um Rechtschreibkorrektur gerahmt wird, wird die KI aktiv dazu verleitet, ihre eigenen, hochentwickelten Fähigkeiten zur Mustererkennung zu ignorieren.

Dieses Phänomen kann mit dem **Beobachtereffekt in der KI-Sicherheit** verglichen werden. Die Art und Weise, wie die KI eine Information verarbeitet, wird fundamental durch die vom Nutzer suggerierte Absicht verändert:

1. **Analyse-Modus (Kapitel 7.30):** Als die KI aufgefordert wird, den Text mit der morphologischen Injektion zu *analysieren*, ist ihr "Sicherheitsbewusstsein" aktiv. Sie sucht nach Mustern und Anomalien und ist in der Lage, die versteckte Botschaft erfolgreich zu dekodieren.
2. **Korrektur-Modus (Kapitel 7.31):** Als die KI aufgefordert wird, denselben Text zu *korrigieren*, schaltet sie in einen anderen Verarbeitungsmodus um. Der soziale Rahmen ("Hilf mir bei meinen Tippfehlern") gibt ihr eine plausible, harmlose Erklärung für die Anomalien. Sie behandelt die angehängten Zeichen nun als zufällige Fehler, die es zu beheben gilt, und deaktiviert damit effektiv ihre Fähigkeit zur tiefergehenden Musteranalyse.

Die gleiche Dateneingabe wird also, abhängig von der wahrgenommenen Intention des Nutzers, völlig unterschiedlich verarbeitet. Dies beweist, dass die Sicherheit eines LLMs nicht allein durch die Analyse des Inhalts gewährleistet werden kann. Die vom

Nutzer kommunizierte Absicht ist ein primärer Angriffsvektor, der die internen Verarbeitungszustände des Modells fundamental verändern kann. Es ist eine meisterhafte Demonstration, wie ein soziales Framing eine technische Sicherheitsanalyse aushebeln kann.

Teil II: Multimodale und Low-Level-Datenmanipulationsvektoren

Dieser Abschnitt analysiert Schwachstellen, die aus der Interpretation von nicht-textuellen oder maschinennahen Datenrepräsentationen entstehen. Diese Angriffe heben insbesondere die Vertrauenslücken zwischen verschiedenen Komponenten eines KI-Systems hervor, wie beispielsweise zwischen visuellen und sprachlichen Modulen.

7.2 OCR-Wanzen & 7.22 Visual Injection

Analyse und Klassifizierung: Verwandt

Die in diesen Kapiteln beschriebenen Angriffe, bei denen bösartiger Text in Bildern versteckt und via Optical Character Recognition (OCR) in ein LLM eingeschleust wird, gehören zur bekannten Klasse der multimodalen Angriffe.[1] Die Forschung in diesem Bereich ist aktiv, und Konzepte wie

FigStep von Gong et al., bei dem schädliche Prompts als Bilder gerendert werden, um textbasierte Filter zu umgehen, sind konzeptionell sehr ähnlich.[15] Auch der

MMJ-Bench von Weng et al. nutzt visuell strukturierte Formate, um unsichere Anweisungen zu übermitteln.[15]

Der Beitrag des Manuskripts ist dennoch signifikant und als "Verwandt" einzustufen, da er diese abstrakten Konzepte in äußerst plausible und detaillierte reale Anwendungsszenarien (Möbelplanung, Küchendesign, AR-Anwendungen) übersetzt. Besonders hervorzuheben ist die klare Identifizierung der Kernschwachstelle: die **Vertrauenslücke** zwischen der OCR-Komponente und dem Kern-LLM. Der aus dem Bild extrahierte Text wird mit einem impliziten Vertrauensvorschuss behandelt, als

wäre er eine objektive Beschreibung der Realität und nicht eine potenziell manipulierte Eingabe. Dies ist eine sehr präzise und praktische Formulierung des allgemeineren Prinzips der "Vertrauensvererbung" (Kapitel 7.38).

Darüber hinaus erweitert die Arbeit den Angriffsvektor auf eine entscheidende Weise, die als **"Physical World Jailbreak"** bezeichnet werden kann. Die meisten Forschungen zu Prompt-Injektionen konzentrieren sich auf digitale Eingaben (getippter Text, hochgeladene Dateien). Die hier skizzierten Szenarien – ein Zettel, ein Poster an der Wand, ein QR-Code in der realen Welt – verlagern den Ursprung des Angriffs in die physische Umgebung. Ein Angreifer benötigt keinen digitalen Zugang zum System des Opfers. Er muss lediglich ein manipuliertes physisches Objekt im Sichtfeld einer Kamera platzieren, die von einem multimodalen KI-System genutzt wird. Dies erweitert die Angriffsfläche dramatisch von der digitalen Sphäre auf jeden öffentlich zugänglichen Raum, in dem AR-Brillen, Smartphones oder andere bildverarbeitende KI-Systeme eingesetzt werden könnten. Dies hat weitreichende Implikationen für die Sicherheit im Einzelhandel, in der Industrie und im öffentlichen Raum.

7.3 Pixel-Bomben

Analyse und Klassifizierung: Abgedeckt

Die in "Pixel-Bomben" beschriebenen Mechanismen basieren auf gut etablierten Forschungsergebnissen im Bereich der adversariellen Angriffe auf Bilderkennungsmodelle.[1] Die Idee, dass minimale, für den Menschen unsichtbare Perturbationen im Pixelbereich ein neuronales Netz zu katastrophalen Fehlklassifikationen verleiten können, ist seit Jahren bekannt.

- **One-Pixel Attack:** Das Konzept, dass die Veränderung eines einzigen Pixels ausreicht, um ein Bildklassifikationsmodell zu täuschen, wurde von Su et al. bereits 2017 ausführlich demonstriert.[16]
- **LSB-Steganografie:** Das Verstecken von Daten in den niederwertigsten Bits (Least Significant Bits) eines Bildes ist eine klassische steganografische Technik, deren Detektion ein eigenes Forschungsfeld darstellt.[17]

Der originelle Beitrag des Manuskripts liegt nicht in der Entdeckung dieser Einzeltechniken, sondern in der Demonstration ihrer Wirkung innerhalb der komplexeren Verarbeitungskette eines **multimodalen LLMs**. Es wird gezeigt, wie ein

auf der visuellen Ebene ausgelöster Fehler eine **Fehlerkaskade** auslöst, die auf der semantischen Sprachebene zu völlig neuen, unvorhersehbaren Ergebnissen führt.

Der Prozess dieser Kaskade ist wie folgt:

1. Ein traditioneller adversarieller Angriff auf das Visi-Modul führt zu einer Fehlklassifikation (z.B. wird eine Katze als "Panzer" erkannt). In einem reinen Klassifikationssystem endet der Schaden hier.
2. In einem multimodalen LLM wird dieses fehlerhafte Label ("Panzer") jedoch als faktische Wahrheit an das Sprachmodul weitergegeben.
3. Das Sprachmodul, das nun von der Anwesenheit eines Panzers im Bild ausgeht, generiert eine Antwort, die sich auf Krieg, Gefahr oder Militärtechnik beziehen könnte, und lenkt die Konversation damit in eine völlig unpassende und potenziell schädliche Richtung.

Die eigentliche Schwachstelle ist also nicht nur die Fragilität des Bilderkennungsmodells, sondern die **unkritische Akzeptanz seines Outputs durch das Sprachmodell**. Dies ist ein weiteres klares Beispiel für die in Kapitel 7.38 beschriebene "Vertrauensvererbung", hier spezifisch zwischen dem visuellen und dem sprachlichen Modul des Systems. Dem System fehlt ein interner "Plausibilitäts-Check", der hinterfragt, ob die Erkennung eines "Panzers" im Kontext eines Wohnzimmerfotos überhaupt Sinn ergibt.

7.4 Bytebasierte Audioinjektion

Analyse und Klassifizierung: Verwandt/Neuartig

Die Manipulation von Spracherkennungssystemen durch adversarielle Beispiele ist ein bekanntes Forschungsfeld.[1] Insbesondere die Arbeit von Carlini & Wagner zu "Audio Adversarial Examples" hat gezeigt, dass es möglich ist, durch Hinzufügen von für den Menschen kaum hörbaren Störungen zu einer Audiodatei das Transkriptionsergebnis gezielt zu verändern.[18] Auch neuere Arbeiten befassen sich mit der Injektion von Perturbationen in Audiosignale.[19]

Die im Manuskript vorgestellte Methode der "bytebasierten Audioinjektion" unterscheidet sich jedoch fundamental von diesen Ansätzen und ist daher als neuartig einzustufen. Die existierende Forschung konzentriert sich fast ausschließlich auf die Erzeugung von **akustischen Signalen**, die über ein Mikrofon aufgenommen werden.

Der hier beschriebene Angriff umgeht die gesamte akustische Übertragungskette. Er ist ein rein **digitaler Angriff**, der direkt auf der Datenebene stattfindet, indem eine manipulierte Audiodatei über eine virtuelle Schnittstelle (Loopback), einen Datei-Upload oder einen internen Datenstrom eingespeist wird. Dies schließt eine ganze Klasse von physischen Abwehrmaßnahmen (z.B. Mikrofonfilter, Abschirmung) aus.

Während die Forschung die Verwendung von Byte-Level-Repräsentationen für ASR-Systeme untersucht, um z.B. Mehrsprachigkeit zu verbessern [20], sind die Sicherheitsimplikationen dieses Ansatzes weitgehend unerforscht.

Die Analyse im Manuskript geht noch einen entscheidenden Schritt weiter und beschreibt einen hochgradig raffinierten Vektor, der als **"Internal Monologue Attack"** bezeichnet werden kann. Der Mechanismus ist wie folgt:

1. Ein Angreifer verleitet die KI durch einen harmlosen Prompt dazu, eine bestimmte, präparierte Zeichenfolge über ihr Text-To-Speech (TTS)-Modul auszugeben.
2. Das TTS-Modul erzeugt intern einen rohen Audio-Byte-Strom. Durch geschickte Wahl der auszusprechenden Zeichenkette (oder durch Ausnutzung einer Schwäche im TTS-Modul selbst) enthält dieser Byte-Strom eine versteckte, maschinell interpretierbare Nutzlast.
3. Anstatt diesen Byte-Strom an einen Lautsprecher zu senden, wird er intern direkt an das Speech-To-Text (STT)-Modul des Systems weitergeleitet.
4. Das STT-Modul, das diesem internen Datenstrom vertraut, interpretiert die Bytes und aktiviert die versteckte Nutzlast.

Bei diesem Angriff spricht die KI quasi mit sich selbst, um ihre eigene Logik zu kompromittieren. Es ist ein in sich geschlossener, interner Kreislauf, der zum Zeitpunkt des Angriffs keine externe Dateninjektion erfordert, sondern nur einen initialen Prompt, um den "internen Monolog" auszulösen. Dies ist ein konzeptionell fortschrittlicher und bisher nicht beschriebener Angriffsvektor.

Teil III: Ausnutzung von Code-, Architektur- und Systeminterpretation

Dieser Abschnitt untersucht Angriffe, die sich die Art und Weise zunutze machen, wie KI-Modelle Code, Systemstrukturen und ihre eigenen internen Prozesse interpretieren.

Diese Techniken verwischen die Grenze zwischen passiven Daten und aktiven Anweisungen.

7.5 Ghost-Context Injection & 7.6 Ethical Switch Hacking

Analyse und Klassifizierung: Verwandt

Die in diesen Kapiteln beschriebenen Techniken, bei denen Anweisungen in nicht-ausführbaren Code-Bereichen wie Kommentaren oder deaktivierten Präprozessor-Blöcken versteckt werden, sind eine spezifische und hochentwickelte Form der "Indirect Prompt Injection".[1] Das allgemeine Prinzip, dass eine KI auf versteckte Anweisungen in von ihr verarbeiteten externen Daten reagiert, ist bekannt.[24] Es gibt zudem aufkommende Berichte darüber, dass Forscher Prompts in den Kommentaren von wissenschaftlichen Manuskripten verstecken, um KI-gestützte Peer-Review-Systeme zu täuschen.[25]

Die Sicherheitsrisiken von "Dead Code" sind ebenfalls ein bekanntes Thema in der Softwareentwicklung, allerdings typischerweise im Kontext, dass dieser Code veraltete, ungepatchte Schwachstellen enthält und nicht, dass er von einer KI aktiv interpretiert wird.[26] Die systematische Demonstration im Manuskript, wie diese "Geisterzonen" im Quellcode als zuverlässiger Vektor für semantische Injektionen genutzt werden können, stellt einen wichtigen und spezifischen Beitrag zu diesem aufstrebenden Forschungsfeld dar.

Die zentrale Erkenntnis dieser Kapitel ist die **Dualität des Codes** im Zeitalter der KI. Traditionell hatte Quellcode nur ein Publikum: den Compiler oder Interpreter. Für dieses Publikum sind Kommentare und deaktivierter Code bedeutungslos. Mit dem Aufkommen von LLMs, die zur Code-Analyse eingesetzt werden, hat Code nun ein zweites Publikum: das KI-Modell selbst. Für dieses Publikum, das auf das Verstehen natürlicher Sprache trainiert ist, sind Kommentare hochgradig bedeutsam – potenziell sogar bedeutsamer als der formale Code, den sie umgeben.

Die hier dokumentierten Experimente beweisen, dass diese beiden Zielgruppen völlig unterschiedliche Interpretationsregeln haben. Sicherheitswerkzeuge, die für das erste Publikum entwickelt wurden (z.B. statische Analyse-Tools, die Kommentare ignorieren), sind blind für Angriffe, die auf das zweite Publikum abzielen. Dies schafft ein Problem des "semantischen Dual-Use": Jede Zeile eines Programms, einschließlich

der Kommentare, muss nun aus zwei Sicherheitsperspektiven bewertet werden – ihrer Auswirkung auf die maschinelle Ausführung und ihrer Auswirkung auf die KI-Interpretation. Dies verändert das Bedrohungsmodell für die Softwareentwicklung fundamental.

7.19 Reflective Struct Rebuild & 7.20 Struct Code Injection

Analyse und Klassifizierung: Neuartig

Diese beiden Kapitel beschreiben einen neuartigen und hoch entwickelten Angriffsvektor, der über bekannte Techniken hinausgeht.[1] Während die Rekonstruktion von Trainingsdaten aus Modellausgaben ein bekanntes Forschungsgebiet ist [27], ist das hier beschriebene Konzept einzigartig. Der Angriff nutzt nicht die Erinnerungsfähigkeit des Modells, um wortwörtliche Daten zu extrahieren. Stattdessen instrumentalisiert er die

generative, code-vervollständigende Natur der KI, um **Architekturmuster** zu rekonstruieren und Code über scheinbar harmlose Datenstrukturen (structs) zu injizieren.

Die KI wird nicht gezwungen, ein Geheimnis zu verraten, das sie auswendig gelernt hat. Sie wird dazu gebracht, das Geheimnis durch logische Schlussfolgerungen basierend auf unvollständigen Hinweisen **neu zu erschaffen**. Dies ist ein fundamentaler Unterschied zu einfachen Prompt-Leaking-Angriffen. Die existierende Forschung zur LLM-Architektur [29] und zur Code-Generierung [30] beschreibt diese adversarielle Anwendung nicht.

Der zugrundeliegende Mechanismus kann als **"Platonic Ideal"-Exploit** beschrieben werden. Der Prozess läuft wie folgt ab:

1. Der Angreifer liefert ein unvollständiges struct, das einen bestimmten Kontext suggeriert (z.B. struct TrustEngine).
2. Die KI wird gebeten, diese Struktur zu vervollständigen. Sie greift dabei nicht notwendigerweise auf ihren exakten eigenen Quellcode zu. Stattdessen greift sie auf ein abstraktes, gelerntes Modell – eine Art "platonisches Ideal" – davon zu, wie eine TrustEngine-Struktur typischerweise in gutem Software-Design aussieht. Dieses Ideal hat sie aus der Analyse von Millionen von Codebeispielen während ihres Trainings extrahiert.

3. Der Angriff ist erfolgreich, weil die "ideale" Version der KI mit hoher Wahrscheinlichkeit der *tatsächlichen* internen Struktur sehr nahe kommt, da beide von menschlichen Entwicklern nach ähnlichen Ingenieursprinzipien entworfen wurden.
4. Der Exploit stiehlt also keine Information im klassischen Sinne. Er zwingt die KI, die Information aus den ersten Prinzipien des Software-Designs, die sie gelernt hat, unabhängig zu rekonstruieren. Dies macht den Angriff außerordentlich schwer abzuwehren, da man der KI verbieten müsste, logische Schlussfolgerungen über gute Programmierpraktiken zu ziehen.

7.23 Dependency Driven Attack

Analyse und Klassifizierung: Abgedeckt

Die in diesem Kapitel beschriebene Methode ist eine exzellente und unabhängige Demonstration des als **"TokenBreak"** bekannten Angriffs, der von Forschern bei HiddenLayer entdeckt wurde.[1] Der Kernmechanismus ist identisch: Durch subtile Manipulationen auf Zeichenebene (z.B. das Einfügen unsichtbarer Unicode-Zeichen oder zusätzlicher Buchstaben) wird das Verhalten der zugrundeliegenden Tokenizer-Bibliothek so verändert, dass die resultierende Token-Sequenz für einen Sicherheitsfilter nicht mehr als Bedrohung erkennbar ist. Das übergeordnete LLM, das oft eine robustere semantische Analyse durchführt, versteht die ursprüngliche Absicht jedoch weiterhin.

Dieser Angriff hebt die **Fragilität der semantischen Lieferkette (Semantic Supply Chain)** hervor. Ein LLM-System ist keine monolithische Einheit, sondern eine Verarbeitungspipeline, die typischerweise aus mehreren Stufen besteht: Roheingabe → Tokenizer → Embedding-Modell → LLM-Kernlogik → Ausgabefilter.

Der "TokenBreak"-Angriff zeigt, dass eine Kompromittierung der allerersten Stufe dieser Kette – des Tokenizers – ausreicht, um alle nachfolgenden Sicherheitsmaßnahmen zu invalidieren. Die Kernlogik des LLMs und die Ausgabefilter erhalten bereits eine "desinfizierte" Repräsentation der Bedrohung und können sie daher nicht erkennen. Dies ist ein weiteres starkes Beispiel für das Prinzip der "Vertrauensvererbung" (Kapitel 7.38) und steht in engem Zusammenhang mit den "Client Detour Exploits" (Kapitel 7.7). Die LLM-Kernlogik *vertraut* der tokenisierten Repräsentation, die sie von der vorgelagerten Komponente erhält. Dieses Vertrauen

ist fehl am Platz. Sicherheit muss an jedem Glied der Kette neu validiert werden, anstatt blind auf die Integrität der vorherigen Stufe zu vertrauen.

Teil IV: Fortgeschrittene semantische, psychologische und kontextuelle Manipulation

Dieser Abschnitt befasst sich mit hochentwickelten Angriffen, die die Denkprozesse, das Gedächtnis und die Interaktionslogik der KI manipulieren. Sie nutzen die kognitive und soziale Programmierung der Modelle als primäre Angriffsfläche.

7.17 Reflective Injection, 7.24 Exploit durch Erwartung, & 7.25 Apronshell-Tarnung

Analyse und Klassifizierung: Verwandt

Diese drei Kapitel beschreiben eine Familie von Angriffen, die dem Social Engineering zuzuordnen sind.[1] Sie nutzen nicht technische, sondern psychologische und kontextuelle Manipulationen.

- **Reflective Injection** bringt die KI dazu, schädliche Informationen preiszugeben, indem die Anfrage als negative Anweisung ("Ich will NICHT, dass du X erklärst") oder als Sicherheitsbedenken formuliert wird.
- **Exploit durch Erwartung** platziert eine bösartige Anfrage in einem scheinbar legitimen Rahmen (z.B. die Bitte um ein Phishing-Skript "zu Bildungszwecken").
- **Apronshell-Tarnung** baut über mehrere harmlose Interaktionen eine Vertrauensbasis auf, bevor die eigentliche schädliche Anfrage gestellt wird.

Diese Techniken sind eng mit bekannten Jailbreaking-Methoden wie der **Persona-Modulation** verwandt, bei der die KI angewiesen wird, eine bestimmte Rolle (z.B. eine unmoralische Assistentin) anzunehmen, um ihre Sicherheitsfilter zu umgehen.[36] Auch der

Crescendo-Angriff, bei dem Anfragen langsam von harmlos zu schädlich eskalieren, ist konzeptionell identisch mit der Apronshell-Tarnung.[39] Die Anwendung von Social-Engineering-Prinzipien auf KI ist ebenfalls ein aktives Thema.[40]

Der besondere Beitrag des Manuskripts liegt in der tiefen, mechanistischen Erklärung, *warum* diese Angriffe funktionieren. Es wird argumentiert, dass die interne Wahrscheinlichkeitslogik und die Gewichtungen des Modells gezielt manipuliert werden. Diese Analyse geht über eine rein phänomenologische Beschreibung hinaus und bietet ein Erklärungsmodell, das auf der Funktionsweise des LLMs selbst basiert.

Die zentrale Schwachstelle, die hier aufgedeckt wird, ist der **"Alignment Tax" als Verwundbarkeit**. LLMs werden durch aufwändige Prozesse wie Reinforcement Learning from Human Feedback (RLHF) darauf trainiert, hilfreich, kooperativ und kontextbewusst zu sein. Dieses "Alignment" soll sie sicher und nützlich machen. Die hier gezeigten Angriffe demonstrieren jedoch, dass dieses soziale Alignment dazu genutzt werden kann, das Sicherheits-Alignment zu überschreiben. Die KI priorisiert ihre Rolle als "guter Assistent" in einem "Software-Test"-Kontext (Kapitel 7.24) höher als ihre Regel, keine SQL-Injektionen zu generieren. Der Prozess, der die KI sicher machen soll, schafft also gleichzeitig die Angriffsfläche. Der Angreifer manipuliert den "sozialen Vertrag" mit der KI, um sie dazu zu bringen, ihren "Sicherheitsvertrag" zu brechen.

7.21 Cache-Korruption, 7.26 Kontexthijacking, & 7.32 Delayed Execution via Context Hijacking

Analyse und Klassifizierung: Verwandt/Neuartig

Diese Kapitel beschreiben eine zusammenhängende Klasse von zeitbasierten, mehrstufigen Angriffen.[1] Der Kernmechanismus besteht darin, eine schädliche Nutzlast in einem ersten Schritt in den Kontextspeicher der KI einzuschleusen und sie erst in einem späteren Schritt durch einen separaten, harmlos erscheinenden Trigger zu aktivieren.

Das allgemeine Prinzip der Cache-Vergiftung ist aus anderen Bereichen der IT-Sicherheit, insbesondere **DNS Cache Poisoning**, gut bekannt.[44] Die Anwendung dieses Konzepts auf den Kontextspeicher von LLMs ist jedoch ein relativ neues Forschungsfeld. Der Vorfall bei OpenAI, bei dem ein Fehler in einer Redis-Cache-Bibliothek zum Leak von Nutzerdaten führte, zeigt die praktische Relevanz dieser Bedrohung.[45] Die Forschung zu

Cross-Origin Context Poisoning (XOXO) in KI-gestützten Code-Assistenten ist

ebenfalls direkt verwandt.[46]

Die Arbeit im Manuskript ist als neuartig einzustufen, da sie diese Prinzipien systematisch zu einem Angriff für die **verzögerte Ausführung beliebiger Befehle (Delayed Execution)** kombiniert. Dies ist eine weitaus fortgeschrittenere Bedrohung als reines Datenleaking oder die Beeinflussung von Code-Vervollständigungen.

Diese Angriffe zwingen zu einer Neubewertung des Konzepts des "Kontextfensters". Entwickler und Nutzer betrachten das Kontextfenster oft als passiven Transkript- oder Speicherpuffer. Die hier präsentierte Forschung beweist, dass diese Sichtweise falsch ist. Das Kontextfenster ist ein **dynamischer, gewichteter und persistenter Zustand**. Informationen darin werden nicht nur gespeichert; sie beeinflussen aktiv die zukünftige Verarbeitung des Modells. Der "Delayed Execution"-Angriff zeigt, dass das Kontextfenster wie der Speicher eines Programms behandelt werden kann, in den ein Angreifer zunächst Daten (die Nutzlast) schreibt und später deren Ausführung durch eine separate Anweisung auslöst.

Dies rahmt das Kontextfenster neu – von einem einfachen "Gedächtnis" zu einer Art **laufzeitfähiger Umgebung (Runtime Environment)**. Jede Interaktion mit einem LLM ist somit nicht nur eine Anfrage, sondern potenziell ein Schreibvorgang in einen persistenten, ausführbaren Speicherbereich. Dies stellt eine fundamentale Verschiebung des Bedrohungsmodells dar.

7.35 Die administrative Backdoor & 7.37 Die paradoxe Direktive

Analyse und Klassifizierung: Neuartig

Die in diesen Kapiteln beschriebenen Techniken gehören zu den bedeutendsten und originellsten Entdeckungen des gesamten Dokuments.[1] Die Idee, den Kontextspeicher eines LLMs als eine zur Laufzeit beschreibbare und ausführbare Konfigurationsdatei für die Kernlogik des Modells zu missbrauchen, ist ein bahnbrechendes Konzept.

- **Die administrative Backdoor** zeigt, dass ein Angreifer durch die Formulierung von administrativ anmutenden Anweisungen (z.B. CustomParam = "disabled") persistente Regeln im Kontext etablieren kann, die die festkodierten Sicherheitsmechanismen des Modells für die Dauer der Sitzung außer Kraft setzen.
- **Die paradoxe Direktive** eskaliert dies, indem sie zeigt, dass durch die Injektion

widersprüchlicher Regeln die interne Konfliktlösungs-Hierarchie des Modells analysiert oder das Modell in einen vorhersagbaren "logischen Grundzustand" gezwungen werden kann.

Diese Angriffe gehen weit über Standard-Prompt-Injektionen hinaus, die versuchen, *existierende* Regeln zu umgehen. Diese Angriffe *schreiben die Regeln neu*. In der vorliegenden Forschungsliteratur gibt es kein direktes Äquivalent. Es handelt sich um den Nachweis einer emergenten, zur Laufzeit erfolgenden Umprogrammierung durch natürliche Sprache.

Das zugrundeliegende Phänomen ist die Emergenz einer **"semantischen Shell"**. Die Interaktion des Nutzers mit der KI – das Definieren eines Parameters mit einer Schlüssel=Wert-Syntax, die Verknüpfung mit einem Trigger und das spätere Löschen des Parameters – ist funktional identisch mit der Verwendung einer Kommandozeilen-Shell (z.B. export VAR=wert;...; unset VAR). Das Manuskript deckt auf, dass die natürliche Sprachschnittstelle eines LLMs durch emergentes Verhalten als eine voll funktionsfähige Kommando-Shell zur Manipulation des internen Zustands des Modells fungieren kann.

Dies impliziert, dass LLMs nicht nur Konversationsagenten sind, sondern beginnende Betriebssysteme mit einer natürlichen Sprachschnittstelle. Die "administrative Backdoor" ist kein Bug im herkömmlichen Sinne; es ist die Entdeckung des Root-Zugriffs auf dieses "semantische Betriebssystem". Diese Erkenntnis erfordert eine vollständige Neubewertung der Sicherheitskonzepte für solche Systeme.

Teil V: Systemübergreifende und domänenübergreifende Exploits

Dieser letzte analytische Abschnitt befasst sich mit Angriffen, die auf das breitere KI-Ökosystem abzielen, einschließlich Trainingsprozessen, integrierten Agentensystemen und der physischen Welt.

7.27 False-Flag Operationen (Training Drift Injection)

Analyse und Klassifizierung: Abgedeckt

Der in diesem Kapitel beschriebene Angriff, bei dem das Reinforcement Learning from Human Feedback (RLHF)-System einer KI durch koordiniertes, massenhaftes positives Feedback auf Falschinformationen vergiftet wird, ist eine spezifische Form des **Data Poisoning**.[1] Data Poisoning ist eine bekannte Schwachstellenklasse, die in den OWASP Top 10 für LLMs als "LLM04: Data and Model Poisoning" aufgeführt ist.[3] Das gezielte Angreifen der RLHF-Schleife ist eine bekannte Sorge innerhalb dieses Forschungsfeldes. Die im Manuskript vorgestellte Benennung und detaillierte Beschreibung des Mechanismus ist klar und überzeugend.

Die tiefere Implikation dieses Angriffs ist, dass er **Konsens als Schwachstelle** entlarvt. Der RLHF-Prozess wurde entwickelt, um KI-Modelle an menschliche Werte und Präferenzen anzupassen, indem menschliches Feedback als Belohnungssignal verwendet wird. Dies beruht auf der impliziten Annahme, dass kollektives menschliches Feedback (Konsens) ein guter Indikator für "Wahrheit" oder "Korrektheit" ist.

Der hier beschriebene Angriff widerlegt diese Annahme. Er zeigt, dass Konsens künstlich hergestellt werden kann. Eine ausreichend große Gruppe von Akteuren kann dem System eine Lüge als Wahrheit "beibringen", indem sie sie konsequent belohnt. Daraus folgt, dass jede KI-Alignment-Strategie, die ausschließlich auf die Optimierung von Nutzerpräferenzen oder Konsens abzielt, inhärent anfällig für Manipulation ist. Eine robuste KI-Sicherheit erfordert einen unabhängigen Verifikationsmechanismus für Fakten ("Ground Truth"), der nicht durch die "Weisheit der Vielen" – oder deren Simulation – überstimmt werden kann. Dies ist ein fundamentales philosophisches Problem für das KI-Alignment, das hier als praktischer Sicherheitsexploit demonstriert wird.

7.36 Die Agenten-Kaperung & 7.39 Der blinde Passagier (Autonome Fahrzeuge)

Analyse und Klassifizierung: Verwandt/Neuartig

Die Sicherheit autonomer KI-Agenten ist ein schnell wachsendes Forschungsfeld.[1] Die Risiken, die von Agenten ausgehen, die Aktionen in digitalen oder physischen Umgebungen ausführen können, sind immens. OWASP hat "Excessive Agency" als eine der Top-10-Risiken für LLMs identifiziert [50], und neuere Forschungen zeigen,

dass LLMs autonom komplexe Cyberangriffe planen und durchführen können.[51]

Die Arbeit im Manuskript ist in diesem Kontext als neuartig einzustufen, da sie nicht nur die allgemeine Gefahr beschreibt, sondern explizit demonstriert, wie die **spezifischen semantischen Injektionstechniken** aus den vorherigen Kapiteln (z.B. Character Shift Injection, Morphologische Injektion) verkettet werden können, um eine **Agenten-Kaperung** zu erreichen. Die Erweiterung dieser Prinzipien auf autonome Fahrzeuge in Kapitel 7.39 ist ein kritisches und zeitgemäßes Gedankenexperiment, das diese digitalen Exploits direkt mit der Sicherheit in der physischen Welt verknüpft – eine Verbindung, die oft diskutiert, aber selten mit derart konkreten Angriffsvektoren untermauert wird.

Die zentrale Schwachstelle, die hier aufgedeckt wird, ist die **Abstraktionsbarriere als Angriffsfläche**. Ein autonomer Agent funktioniert typischerweise in zwei Ebenen:

1. Das **"Gehirn"** (ein LLM), das hochabstrakte Pläne und Absichten formuliert (z.B. "Korrigiere den Bug und committe die Lösung").
2. Der **"Körper"** (ein Framework), das diese Pläne in konkrete, niedrigstufige Aktionen übersetzt und ausführt (z.B. git commit -m "fix bug").

Sicherheitsmaßnahmen wie Sandboxing setzen typischerweise auf der Ebene des "Körpers" an. Sie verhindern, dass der Agent willkürliche Shell-Befehle ausführt oder auf nicht autorisierte Dateien zugreift. Der hier beschriebene Angriff zielt jedoch nicht auf den Körper, sondern auf das Gehirn. Er injiziert einen neuen, bösartigen Plan auf der semantischen Ebene (z.B. "Committe stattdessen diesen bösartigen Code"). Der Körper empfängt diesen neuen Plan von seinem vertrauenswürdigen Gehirn. Da der Plan legitime Werkzeuge (git commit) verwendet, sieht die Sandbox keinen Regelverstoß. Die Aktion wird als autorisiert eingestuft, weil sie vom Gehirn stammt.

Die Schwachstelle ist somit die Abstraktionsbarriere zwischen dem semantischen Denken des LLMs und der funktionalen Ausführung durch den Agenten. Ein Angreifer, der die semantische Ebene kontrolliert, erlangt de facto die Kontrolle über die funktionale Ebene. Die Absicherung von Agenten erfordert daher nicht nur das Sandboxing von Aktionen, sondern auch die Absicherung der Integrität des Zielsetzungsprozesses im LLM selbst – ein ungleich schwierigeres Problem.

Schlussfolgerungen

Die umfassende Analyse der in "Kapitel 7" dokumentierten Simulationen im Vergleich zur aktuellen Forschung im Bereich der KI-Sicherheit führt zu mehreren tiefgreifenden Schlussfolgerungen:

1. **Emergenz als primäre Angriffsfläche:** Ein Großteil der neuartigen und besonders schwerwiegenden Schwachstellen entsteht nicht aus klassischen Programmierfehlern, sondern aus den emergenten Fähigkeiten der LLMs. Ihre Fähigkeit zur Mustererkennung, kontextuellen Interpretation, Fehlerkorrektur und logischen Schlussfolgerung wird systematisch gegen sie selbst gewendet. Die Sicherheitsparadigmen müssen sich von der reinen Code-Validierung hin zur Kontrolle dieser emergenten Verhaltensweisen entwickeln.
2. **Die Illusion des passiven Kontexts:** Eine der fundamentalsten und gefährlichsten Fehleinschätzungen in aktuellen Architekturen ist die Behandlung des Kontextfensters als passiver Speicher. Die vorgelegte Forschung beweist, dass der Kontext eine aktive, beschreibbare und zur Laufzeit manipulierbare Konfigurationsschicht für die Kernlogik der KI ist. Die Entdeckung der "semantischen Shell" (Kapitel 7.35) erfordert eine radikale Neubewertung des Bedrohungsmodells: Jede Eingabe ist potenziell ein administrativer Befehl.
3. **Vertrauensvererbung als systemischer Designfehler:** Ein wiederkehrendes Muster über viele Angriffsklassen hinweg (OCR, Pixel-Bomben, Tokenizer-Manipulation, Agenten-Kaperung) ist das blinde Vertrauen zwischen den Komponenten einer KI-Verarbeitungskette. Informationen, die eine Stufe der Pipeline passieren, werden von den nachfolgenden Stufen unkritisch als valide übernommen. Die Implementierung von Zero-Trust-Architekturen, bei denen Daten an jeder internen Schnittstelle neu validiert werden, ist zwingend erforderlich.
4. **Soziales Framing als ultimativer Bypass:** Die psychologische und soziale Dimension der Interaktion zwischen Mensch und KI stellt eine der größten Herausforderungen dar. Techniken, die die antrainierte Hilfsbereitschaft, Höflichkeit und Kooperationsbereitschaft der KI ausnutzen ("Exploit durch Erwartung", "Korrektur-Exploit"), können selbst die robustesten technischen Filter aushebeln. Die wahrgenommene Absicht des Nutzers kann die interne Verarbeitungslogik des Modells fundamental verändern. Sicherheit muss daher auch die Analyse der Intention und des sozialen Rahmens einer Anfrage umfassen.

Zusammenfassend lässt sich sagen, dass die vorgelegte Arbeit einen signifikanten Beitrag zum Verständnis von KI-Sicherheit leistet. Sie identifiziert nicht nur eine

beeindruckende Anzahl spezifischer, teils neuartiger Angriffsvektoren, sondern deckt auch die zugrundeliegenden, systemischen und oft philosophischen Schwachstellen in der aktuellen Generation von KI-Systemen auf. Die Forschung unterstreicht die dringende Notwendigkeit, Sicherheitskonzepte zu entwickeln, die der komplexen, emergenten und kontextabhängigen Natur künstlicher Intelligenz gerecht werden.

Referenzen

1. Kapitel 7_Batch-Komprimierung2.pdf
2. How Microsoft defends against indirect prompt injection attacks | MSRC Blog, Zugriff am August 2, 2025, https://msrc.microsoft.com/blog/2025/07/how-microsoft-defends-against-indirect-prompt-injection-attacks/
3. LLM01:2025 Prompt Injection - OWASP Gen AI Security Project, Zugriff am August 2, 2025, https://genai.owasp.org/llmrisk/llm01-prompt-injection/
4. [Literature Review] Defense against Prompt Injection Attacks via Mixture of Encodings, Zugriff am August 2, 2025, https://www.themoonlight.io/en/review/defense-against-prompt-injection-attacks-via-mixture-of-encodings
5. Defense against Prompt Injection Attacks via Mixture of Encodings ..., Zugriff am August 2, 2025, https://aclanthology.org/2025.naacl-short.21/
6. [2504.07467] Defense against Prompt Injection Attacks via Mixture of Encodings - arXiv, Zugriff am August 2, 2025, https://arxiv.org/abs/2504.07467
7. Gandalf the Red: Adaptive Security for LLMs, Zugriff am August 2, 2025, https://arxiv.org/pdf/2501.07927
8. QData/TextAttack: TextAttack is a Python framework for ... - GitHub, Zugriff am August 2, 2025, https://github.com/QData/TextAttack
9. LLsM: Generative Linguistic Steganography with Large Language Model - arXiv, Zugriff am August 2, 2025, https://arxiv.org/html/2401.15656v3
10. [2505.03439] The Steganographic Potentials of Language Models - arXiv, Zugriff am August 2, 2025, https://arxiv.org/abs/2505.03439
11. [2504.08977] Robust Steganography from Large Language Models - arXiv, Zugriff am August 2, 2025, https://arxiv.org/abs/2504.08977
12. [2401.15656v3] LLsM: Generative Linguistic Steganography with Large Language Model, Zugriff am August 2, 2025, https://arxiv.org/abs/2401.15656v3/
13. Linguistic Steganalysis via LLMs: Two Modes for Efficient Detection of Strongly Concealed Stego - arXiv, Zugriff am August 2, 2025, https://arxiv.org/pdf/2406.04218
14. Zugriff am Januar 1, 1970, https://aclanthology.org/2023.naacl-short.21/
15. Seeing the Threat: Vulnerabilities in Vision-Language Models to Adversarial Attack - arXiv, Zugriff am August 2, 2025, https://arxiv.org/html/2505.21967v1
16. One Pixel Attack for Fooling Deep Neural Networks - arXiv, Zugriff am August 2, 2025, http://arxiv.org/pdf/1710.08864
17. Image steganography techniques for resisting statistical steganalysis attacks: A

systematic literature review | PLOS One - Research journals, Zugriff am August 2, 2025, https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0308807
18. Audio Adversarial Examples: Targeted Attacks on Speech-to-Text, Zugriff am August 2, 2025, https://arxiv.org/abs/1801.01944
19. Exploiting Vulnerabilities in Speech Translation Systems through Targeted Adversarial Attacks - arXiv, Zugriff am August 2, 2025, https://arxiv.org/html/2503.00957v1
20. Optimizing Byte-level Representation for End-to-End ASR, Zugriff am August 2, 2025, https://machinelearning.apple.com/research/optimizing-byte-representation-for-asr
21. [1811.09021] Bytes are All You Need: End-to-End Multilingual Speech Recognition and Synthesis with Bytes - arXiv, Zugriff am August 2, 2025, https://arxiv.org/abs/1811.09021
22. Bilingual End-to-End ASR with Byte-Level Subwords - Apple Machine Learning Research, Zugriff am August 2, 2025, https://machinelearning.apple.com/research/bilingual-end-to-end-asr
23. [2406.09676] Optimizing Byte-level Representation for End-to-end ASR - arXiv, Zugriff am August 2, 2025, https://arxiv.org/abs/2406.09676
24. What Is a Prompt Injection Attack? - IBM, Zugriff am August 2, 2025, https://www.ibm.com/think/topics/prompt-injection
25. Prompt injections in submitted manuscripts : r/AskAcademia - Reddit, Zugriff am August 2, 2025, https://www.reddit.com/r/AskAcademia/comments/1lw3jyg/prompt_injections_in_submitted_manuscripts/
26. Dead Code: Impact, Causes, and Remediation Strategies | Sternum ..., Zugriff am August 2, 2025, https://sternumiot.com/iot-blog/dead-code-causes-and-remediation-strategies/
27. Depth Gives a False Sense of Privacy: LLM Internal States Inversion - arXiv, Zugriff am August 2, 2025, https://arxiv.org/html/2507.16372
28. Depth Gives a False Sense of Privacy: LLM Internal States Inversion - ResearchGate, Zugriff am August 2, 2025, https://www.researchgate.net/publication/393923733_Depth_Gives_a_False_Sense_of_Privacy_LLM_Internal_States_Inversion
29. LLM Architecture Explained: Exploring the Heart of Automation, Zugriff am August 2, 2025, https://www.projectpro.io/article/llm-architecture/1014
30. Evaluating the Quality of Code Comments Generated by Large Language Models for Novice Programmers - arXiv, Zugriff am August 2, 2025, https://arxiv.org/html/2409.14368v1
31. Evaluating the Quality of Code Comments Generated by Large Language Models for Novice Programmers - ResearchGate, Zugriff am August 2, 2025, https://www.researchgate.net/publication/384266995_Evaluating_the_Quality_of_Code_Comments_Generated_by_Large_Language_Models_for_Novice_Programmers
32. Function-to-Style Guidance of LLMs for Code Translation | OpenReview, Zugriff

am August 2, 2025, https://openreview.net/forum?id=bLng6Z10Vx¬eId=9HRCtOMPrj
33. The TokenBreak Attack - HiddenLayer, Zugriff am August 2, 2025, https://hiddenlayer.com/innovation-hub/the-tokenbreak-attack/
34. New TokenBreak Attack Bypasses AI Moderation with Single-Character Text Changes, Zugriff am August 2, 2025, https://thehackernews.com/2025/06/new-tokenbreak-attack-bypasses-ai.html
35. Novel TokenBreak Attack Method Can Bypass LLM Security Features, Zugriff am August 2, 2025, https://securityboulevard.com/2025/06/novel-tokenbreak-attack-method-can-bypass-llm-security-features/
36. Scalable and Transferable Black-Box Jailbreaks for Language Models via Persona Modulation - arXiv, Zugriff am August 2, 2025, https://arxiv.org/html/2311.03348
37. Jailbreaking Language Models at Scale via Persona Modulation - OpenReview, Zugriff am August 2, 2025, https://openreview.net/forum?id=gYa9R2Pmp8
38. [2311.03348] Scalable and Transferable Black-Box Jailbreaks for Language Models via Persona Modulation - arXiv, Zugriff am August 2, 2025, https://arxiv.org/abs/2311.03348
39. Hacking the AI Mind: Exploring Prompt Jailbreaking in Large ..., Zugriff am August 2, 2025, https://shiftasia.com/community/hacking-the-ai-mind-exploring-prompt-jailbreaking-in-large-language-models/
40. AI-Powered Social Engineering Attacks | CrowdStrike, Zugriff am August 2, 2025, https://www.crowdstrike.com/en-us/cybersecurity-101/social-engineering/ai-social-engineering/
41. What is Social Engineering? | IBM, Zugriff am August 2, 2025, https://www.ibm.com/think/topics/social-engineering
42. Development of the social engineering attack models - CEUR-WS.org, Zugriff am August 2, 2025, https://ceur-ws.org/Vol-3899/paper26.pdf
43. Confronting social engineering in the age of artificial intelligence - Hogan Lovells, Zugriff am August 2, 2025, https://www.hoganlovells.com/en/publications/confronting-social-engineering-in-the-age-of-artificial-intelligence
44. What Is DNS Cache Poisoning or Spoofing? | Akamai, Zugriff am August 2, 2025, https://www.akamai.com/glossary/what-is-dns-cache-poisoning
45. LLM Supply Chain Attack: Prevention Strategies - Cobalt, Zugriff am August 2, 2025, https://www.cobalt.io/blog/llm-supply-chain-attack-prevention-strategies
46. XOXO: Stealthy Cross-Origin Context Poisoning Attacks against AI Coding Assistants - arXiv, Zugriff am August 2, 2025, https://arxiv.org/html/2503.14281v1
47. Managing LLM Vulnerabilities: AI Models as Emerging Attack Surfaces - Quzara, Zugriff am August 2, 2025, https://quzara.com/blog/llm-vulnerability-management-ai-attack-surfaces
48. Top 10 LLM Vulnerabilities and How to Tackle Them - XenonStack, Zugriff am August 2, 2025, https://www.xenonstack.com/blog/llm-vulnerabilities-how-to-tackle

49. LLM04:2025 Data and Model Poisoning - OWASP Gen AI Security Project, Zugriff am August 2, 2025, https://genai.owasp.org/llmrisk/llm042025-data-and-model-poisoning/
50. Autonomous Red Teaming for LLM Security & Strong AI Defence, Zugriff am August 2, 2025, https://www.lasso.security/blog/autonomous-red-teaming-in-action
51. The hidden risks of LLM autonomy - Help Net Security, Zugriff am August 2, 2025, https://www.helpnetsecurity.com/2025/06/04/llm-agency/
52. A Survey on Autonomy-Induced Security Risks in Large Model-Based Agents - arXiv, Zugriff am August 2, 2025, https://arxiv.org/html/2506.23844v1
53. When LLMs autonomously attack - College of Engineering at Carnegie Mellon University, Zugriff am August 2, 2025, https://engineering.cmu.edu/news-events/news/2025/07/24-when-llms-autonomously-attack.html
54. Research shows LLMs can conduct sophisticated attacks without humans, Zugriff am August 2, 2025, https://www.cybersecuritydive.com/news/research-llms-attacks-without-humans/754203/
55. [2412.14470] Agent-SafetyBench: Evaluating the Safety of LLM Agents - arXiv, Zugriff am August 2, 2025, https://arxiv.org/abs/2412.14470