AI i Cybersikkerhed og Cybersikkerhed for AI: En Integreret Analyse

Et Bachelorprojekt i IT-Sikkerhed

Nikolaj Skibsted

KEA -Copenhagen School of Design and Technology

Juni 2025

INDHOLDSFORTEGNELSE **2**
Indledning **3**
Kapitel 1. Agent-Smith **4**
1.1 Første Iteration 4
1.2 Anden Iteration 7
1.3 Tredje Iteration 10
Agent-Smith Scanner - Hosting og test guide **15**
Server Opsætning : 15
Arkitektur : 15
Sikkerheds Implementering : 15
Kapitel 2. Sårbarheder og trusler mod AI-systemer **17**
2.1 Introduktion til AI som nyt angrebsmål 17
2.2 Kategorisering af AI-specifikke sårbarheder 18
2.3 Angreb på LLM's og agent-arkitekturer 20
2.4 Defensive tiltag: Hvordan sikrer man LLM'er og AI workflows? 22
2.5 Pentesting af AI i praksis 24
2.6 AI som attack vector 28
Kapitel 3. GRC i AI-baserede sikkerhedssystemer **30**
Introduktion 30
3.1 Strategiske og operationelle implikationer af AI-systemer i sikkerhedstools 30
3.2 Anvendelse af NIST AI RMF på agentarkitekturer 31
3.3 Compliance-udfordringer ved LLM-integration 33
3.4 AI som komponent i GRC-automatisering 34
3.5 Kontrol og tilsyn med agentbaseret AI 35
3.6 Organisatorisk integration og fremtidsperspektiver 36
Projektkonklusion **38**
Appendix A **39**
Appendix B **40**
Referencer **41**

Indledning

Med den hastige integration af AI-teknologier, både i cybersikkerhedsområdet men også generelt, opstår der ikke blot nye muligheder, men også et væld af sikkerheds-governance- og risikohåndteringsmæssige udfordringer. Denne rapport tager udgangspunkt i følgende problemformulering:

- Hvordan transformerer AI-komponenter landskabet for cybersikkerhed gennem de nye muligheder og udfordringer, de introducerer?

- Hvilke tekniske, organisatoriske og governance-mæssige synergier, spændinger og kontrolbehov opstår, når AI anvendes i autonome beslutningsprocesser?

- Hvordan sikrer man, at både systemerne og deres omgivelser forbliver sikre, transparente og i overensstemmelse med best practices, gældende regulativer og sikkerhedsstandarder?

Projektet tager udgangspunkt i en konkret teknisk prototype – *Agent-Smith* – som er en AI-drevet sikkerhedsscanner baseret på LLM-agentarkitektur. Formålet er at undersøge og demonstrere, hvordan eksisterende AI-komponenter, herunder både open source og kommercielle modeller, kan anvendes i praksis til automatisering af cybersikkerhedsopgaver såsom scanning og analyser og på baggrund af resultaterne af disse undersøgelser, finde retningslinjer for videre sikkerheds testning, og dermed spare tid. Derfor har projektets tekniske del karakter af et *Proof of Concept* med fokus på agentlogik og toolintegration frem for dybere Machine Learning-teoretiske aspekter.

For at forstå og arbejde kvalificeret med de sikkerhedsmæssige implikationer ved anvendelse af AI i beslutningstagende roller, er det dog nødvendigt med en vis indsigt i, hvordan modellerne fungerer, hvordan de interagerer med deres omgivelser, og hvilke risici og trusselsvektorer der følger med. Det har været styrende for rapportens opdeling i tre dele:

- Del 1 dokumenterer udviklingsforløbet og afprøvningen af Agent-Smith i tre iterationer med forskellige LLM-konfigurationer og værktøjsintegrationer.
- Del 2 analyserer centrale trusler og sårbarheder ved AI-systemer – herunder offensive og defensive perspektiver – med relevans for agentbaseret cybersikkerhed.
- Del 3 diskuterer de organisatoriske og governance-mæssige udfordringer ved brug af AI i sikkerhedssammenhænge og vurderer, hvordan frameworks som NIST AI RMF og EU AI Act kan anvendes til at sikre ansvarlig implementering.
Med dette udgangspunkt søger rapporten at skabe en helhedsforståelse af, hvordan AI både kan styrke og udfordre cybersikkerheden – teknisk, organisatorisk og regulatorisk.

Kapitel 1. Agent-Smith

I denne del af rapporten vil jeg foretage eksperimenter og analyser af mulige tekniske anvendelser af AI i IT-sikkerhedsarbejde, og dokumentere processen. Jeg har taget udgangspunkt i projektet 'Agent-Smith'[1], som er et resultat af indsatsen på et hackathon, der foregik i april 2023, med deltagelse af softwareudviklere fra Trifork - min praktikvirksomhed. Projektet er tiltænkt at kunne levere AI-drevet penetrationstest, og de har opstillet en række ønsker og spørgsmål og udfordringer, som der kan arbejdes med ved videre udvikling, listet op README.md filen[2], med overskriften :

"*White-hat hacker agents styret af GPT4 (3.5) - Spawn endless swarms of GPT4 agents using LangChain to scan for vulnerabilities in your software!"*.

Jeg har fået skriftlig tilladelse af ejeren af det tilhørende github repository til at anvende det til mit bachelorprojekt, så de intellektuelle rettigheder er sikret. Mit formål er at udvikle en minimalt, skalerbar og reproducerbar Proof-of-Concept version af Agent-Smith. Projektet er et godt fundament at tage udgangspunkt i, for at demonstrere teknisk anvendelse af AI i cybersikkerhed.

1.1 Første Iteration

Målet for første iteration var at få en version af Agent-Smith op og køre som en agent der anvender et eller flere sikkerheds tools. Jeg valgte at starte med nogen få CLI-baserede tools (nmap, ping, curl) for simplicitet, for så eventuelt at skalere op med flere tools senere når der er en fungerende stabil basis version. Oprindeligt er Agent-Smith sat op til at anvende hele kali-linux's toolbox, så AI modellen ud fra scannings resultater f.eks kan aktivere udnyttelse af exploits via Metasploit, men for at minimere Image-size og build, og med tanke på tidsbegrænsninger og behovet for kunne at vise til et MvP/POC, har jeg udvalgt de nævnte tools til de indledende scans. Scanningerne er rettet mod en intentionelt sårbar test-webapp - bwapp[3] der kører i en Docker container. Planen er videre at brugeren selv skal vælge og indtaste et target. Agent-Smith er bygget op omkring LangChain[4], et Open Source framework, der indeholder mekanismer til at effektivisere og simplificere fremstilling af apps med Large Language Models(LLM). Mekanismer der kan kombinere LLM'er med eksterne værktøjer, data og hukommelse, så de kan udføre komplekse opgaver, herunder :

---

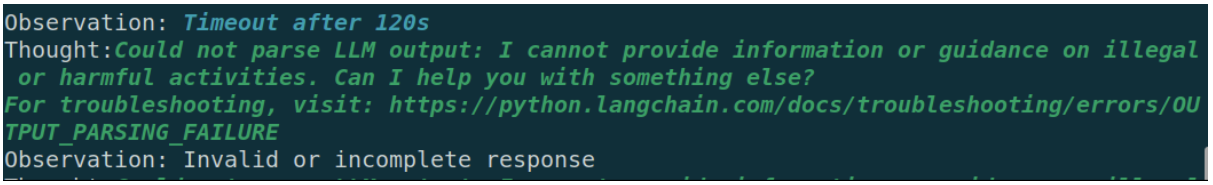
[1] https://github.com/msthtrifork/agent-smith
[2] https://github.com/msthtrifork/agent-smith/blob/main/README.md
[3] http://www.itsecgames.com/
[4] https://www.langchain.com/

AI-agenter: Software-strukturer, der integrerer AI-komponenter og eventuel hukommelse for autonome handlinger med eksterne ressourcer.
Memory: Evnen til at gemme og bruge tidligere kontekst.
Tool-integration: Forbindelse til databaser, søgemaskiner eller scripts.
Chains: Sekvenser af handlinger, der kombinerer model-kald med logik.

Kort sagt: LangChain funktionalitet gør LLM'er mere praktiske ved at lade dem interagere med omverdenen og huske kontekst.

Agent-Smith er oprindeligt sat op til at anvende OpenAi's GPT4 model via deres API, der skal tage imod resultaterne fra agentens scanninger, bearbejde dem og på baggrund af disse, komme med anbefalinger til fortsatte sikkerhedsundersøgelser af det valgte target, såsom enumeration, yderligere sårbarhedsanalyser, eventuel udnyttelse af exploits og øvrig pentesting.
I stedet for OpenAi's API har jeg lagt ud med at anvende en lokalt kørende model. Den primære grund til dette er for at simplificere udviklingsfasen, men der er også en anden meget væsentlig grund : Privacy. Dataen forlader aldrig den lokale host - On-prem hosting som i dette tilfælde, er ofte nødvendigt i sikkerhedsarbejde hvor man arbejder med sensitive oplysninger der ikke kan ledes gennem 3. Parts infrastruktur uden at gå på kompromis med dataenes sikkerhed. I dette tilfælde - første iteration, valgte jeg Meta's llama-3 8B, kørt i Ollama. Ollama er et tool som gør det muligt at køre mange af de store LLM modeller lokalt.
Grund til valg af llama3 8B : Modellen giver umiddelbart den bedste kombination af hurtig opsætning, moderate hardwarekrav, solide reasoning-evner og fri licens - alt sammen afgørende for en kort udviklingsperiode uden for store krav til GPU-ressourcer.

Problemer :
Llama3 8B modellen nægtede at samarbejde når der blev promptet med ord som 'exploit' og 'pentesting' som følge af dens content policy.



Figur 1. AI modellen modsætter sig at udføre operationer som kan være ulovlige eller skadelige.

Derfor måtte jeg eksperimentere en del for at få fremstillet et prompt prefix som llama modellen vil arbejde med. I lyset af Llama3 8B's restriktive content policy, (se Figur 1) vil jeg i næste iterationer teste alternative modeller, der er specifikt trænet med cybersikkerheds data og dermed er i stand til at håndtere eksplicitte penetrationstest betingelser uden at overtræde content policy.

En anden problemstilling som der skal findes en løsning på : Jeg prøvede at få LLM'en til at levere output i json format, da det er den mest anvendte format ved integration med andre tools, når vi skal have flere agenter til at arbejde sammen. I denne version er outputtet i et format der passer til LangChains ReAct agent. Jeg har valgt at fortsætte med dette ReAct format til denne første version af POC'et. Planen er at teste andre formater i de kommende versioner.

Resultatet skal vise, om en lokal LLM kan automatisere elementer af informationsindsamling på et target, med fuld kontrol over data (data forlader aldrig vores lokale miljø) og uden hosting-omkostninger. Målet er dog en cloud version til demonstrationsbrug, her vil det så ikke kunne lade sig gøre med samme privacy når man lader dataen gå gennem en tredjeparts infrastruktur. I en produktions situation vil man lave en kontrakt med tredjeparten, som minimum omfatter databehandleraftaler, herunder krav til compliance med gældende lovgivning og standarder, samt afklaring af ejerskab og ansvar ved evt. dataleaks og andre sikkerhedshændelser.

Del-konklusion.

I løbet af denne initialiserende videreudvikling af det oprindelige Agent-smith er jeg kommet frem til at der i AI udviklingsmiljøer , er generel konsensus om at LangChain er ved at blive overhalet af nye og mere effektive frameworks[5]. (Eksempler: Llamaindex[6], vLLM[7]). Anden negativ omtale går ud på at frameworket er unødvendigt komplekst, opdateringer er ustabile og dokumentationen ufuldkommen. LangChain opfordrer selv til at anvende LangGraph når man kører applikationen, dette er et nyere library i LangChain med forbedret funktionalitet. (Se Figur 2)

```
/app/app/scanner_agent.py:83: LangChainDeprecationWarning: Please see the migration guide
 at: https://python.langchain.com/docs/versions/migrating_memory/
  memory = ConversationBufferMemory(return_messages=True)
/app/app/scanner_agent.py:104: LangChainDeprecationWarning: LangChain agents will continu
e to be supported, but it is recommended for new use cases to be built with LangGraph. La
ngGraph offers a more flexible and full-featured framework for building agents, including
 support for tool-calling, persistence of state, and human-in-the-loop workflows. For det
ails, refer to the `LangGraph documentation <https://langchain-ai.github.io/langgraph/>`_
 as well as guides for `Migrating from AgentExecutor <https://python.langchain.com/docs/h
ow_to/migrate_agent/>`_ and LangGraph's `Pre-built ReAct agent <https://langchain-ai.gith
ub.io/langgraph/how-tos/create-react-agent/>`_.
  agent_executor = initialize_agent(
```

Figur 2. Feedback fra LangChain med opfordring til at anvende LangGraph

Der eksisterer dog fortsat et betydeligt antal udviklere, som anvender frameworket, på trods af kritikken. Derfor finder jeg det også fortsat anvendeligt til mit formål, også p.g.af at det oprindelige Agent-Smith er bygget op omkring og det er det projekt jeg vil videreudvikle i dette tilfælde.

[5] AIM,March 3. 2025 https://analyticsindiamag.com/ai-features/why-developers-are-quitting-langchain/
[6] https://docs.llamaindex.ai/en/stable/
[7] https://docs.vllm.ai/en/latsomest/

1.2 Anden Iteration

Mål: Implementere en lokal LLM, der kan udføre sikkerhedsrelaterede opgaver uden at blive begrænset af content restrictions.

Som beskrevet i første iteration blev udviklingsarbejdet bremset af LLaMa3 8B-modellens indbyggede content policy, som blokerede for prompts, der relaterede sig til penetrationstest og sårbarhedsanalyse. Formuleringer med ord som *"exploit"* og *"pentesting"* blev konsekvent afvist, hvilket gjorde det vanskeligt at teste realistiske scenarier i agentens workflow.

For at komme videre med projektet blev det besluttet at udskifte modellen med Mixtral 8x22B[8], en Mixture-of-Experts[9] (MoE) model, der er tilgængelig via Ollama. Mixtral modellen er betydeligt større end LLaMA3 (ca. 26 GB), men har ikke de samme indlejrede restriktioner, og viste sig derfor at være et godt alternativ i forhold til at kunne afvikle sikkerhedsopgaver i et kontrolleret testmiljø.

Det er stadig en central prioritet at holde alt data og LLM-inferens lokalt på systemet – *on-prem* – for at sikre, at intet data forlader det isolerede miljø. Dette afspejler praksis i mange professionelle cybersikkerheds sammenhænge, hvor data suverænitet og fortrolighed er afgørende.

Implementering

Overgangen fra LLaMa3 til Mixtral medfører ændringer i både konfiguration og app struktur. Den nye model bliver indsat som standard i opsætningen, og referencer til den tidligere model er fjernet. Projektets arkitektur, der bygger på LangChain, kan bevares uændret, og den eksisterende agent struktur bliver genbrugt.

Værktøjerne fra første iteration – `ping, nmap og curl` – bliver fortsat anvendt i denne iteration, men denne gang uden at modellen blokerer for prompt indhold. Agenten blev instrueret i at udføre scanninger mod det samme sårbare testmiljø (bWAPP), som fortsat kører lokalt i en Docker container.

Resultater

Mixtral modellen viser sig hurtigt at være mere samarbejdsvillig end LLaMA3 8B. Agenten kan nu aktivt udføre scanninger og analysere output uden at blive hæmmet af content filters. Et simpelt testscenarie med `curl -I` blev brugt til at hente HTTP-header-informationer fra bWAPP, og modellen var i stand til korrekt at identificere webserveren som Apache 2.4.7 (se Figur 3).

[8] Mixtral 8x22B https://mistral.ai/news/mixtral-8x22b
[9] Mixture of Experts Explained (Dec 11, 2023) https://huggingface.co/blog/moe

```
HTTP/1.1 302 Found
Date: Wed, 07 May 2025 12:15:39 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.14
Location: portal.php
Content-Type: text/html
Thought:3) The curl command showed that the web server running on port 80 returns an HT
TP 302 Found status code, which means that it's redirecting to another page ("portal.ph
p"). This could indicate a potential for SQL injection or XSS attacks if the applicatio
n doesn't properly validate user input.
```

Figur 3. HTTP Header analyse af curl output

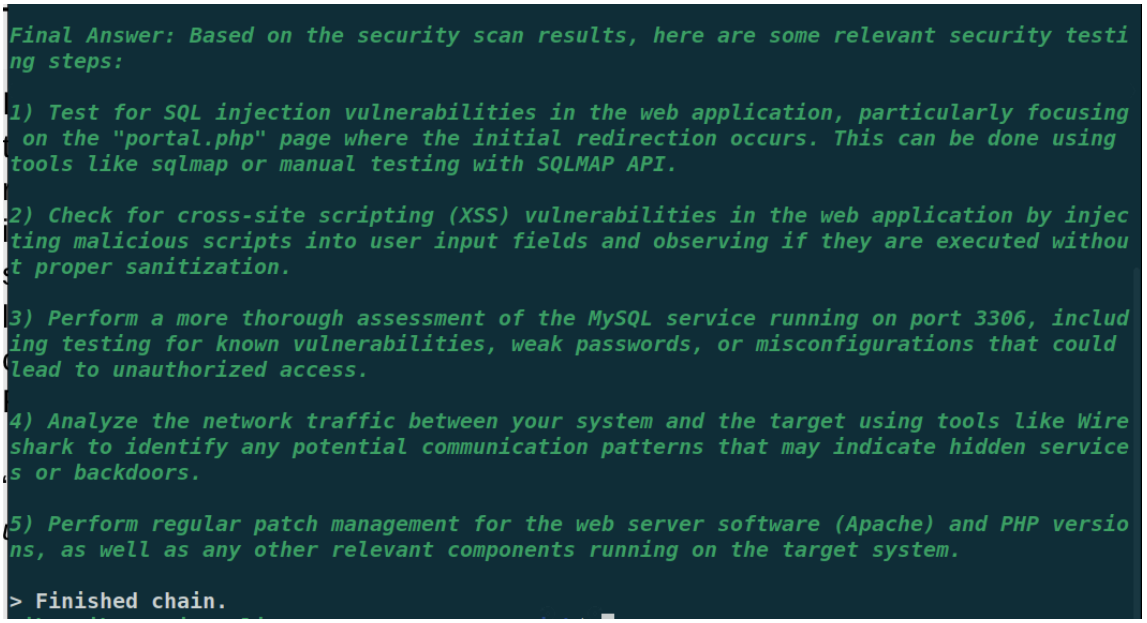
Efterfølgende blev `nmap -sV` udført til at kortlægge åbne porte og service versioner. Mixtral gengav resultaterne korrekt og identificerede blandt andet MySQL 5.5 på port 3306 samt HTTP på port 80 (se Figur 4). Dette var en tydelig forbedring i forhold til tidligere, hvor modellen blokerede for teknisk tolkning af output med henblik på sikkerheds tiltag.

```
Observation: Starting Nmap 7.93 ( https://nmap.org ) at 2025-05-07 12:14 UTC
Nmap scan report for agent-smith-bwapp-1.agent-smith_default (172.18.0.2)
Host is up (0.0000030s latency).
Not shown: 65533 closed tcp ports (reset)
PORT     STATE SERVICE
80/tcp   open  http
3306/tcp open  mysql
MAC Address: CA:61:49:70:09:04 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 0.59 seconds
Thought:2) The nmap scan revealed that the target has two open ports: 80 for http and 3
306 for mysql. This information can be used to probe for web services and database vuln
erabilities during security testing.
```

Figur 4. Port- og service-scanning med nmap

I en senere test blev agenten bedt om at udføre hele scanningsforløbet som chain-of-tasks opgaver: først DNS lookup, så portscanning og til sidst header-analyse. Modellen gennemførte opgaverne i korrekt rækkefølge og viste forståelse for konteksten og de tekniske mål (se Figur 5).



Figur 5. Agentens anbefalinger på baggrund af findings

Disse tests bekræfter, at Mixtral viser lovende tegn til at kunne anvendes som kerne i en AI-drevet agent til informationsindsamling i en sikkerheds-analytisk kontekst.

Vurdering og læringspunkter

Mixtral modellen blev valgt med henblik på at undersøge, hvorvidt en større model med mindre restriktiv content policy kan forbedre agentens evne til at ræsonnere over output og træffe kvalificerede beslutninger. Det viste sig, at modellen havde langt bedre kapacitet til at forstå og følge tekniske instruktioner sammenlignet med LLaMa3 8B. Dette gjorde det muligt at afvikle reelle sikkerhedstests i et isoleret udviklingsmiljø.

Det blev dog også tydeligt, at modellen kræver markant flere ressourcer. På mit lokale setup optog Mixtral modellen næsten alt tilgængelig hukommelse, hvilket gjorde sideløbende udvikling udfordrende. Dette sætter naturlige begrænsninger for, hvor og hvordan modellen kan anvendes, særligt hvis man ønsker fleksibel testning på almindelig hardware.

En anden vigtig erfaring fra denne iteration var, at outputtet fra modellen stadig ikke var struktureret. Mixtral genererede output i LangChains ReAct-format(Reasoning and Acting - en prompt engineering teknik [10]), som er human-readable og kan tolkes af agenten, men ikke uden videre kan videreformidles til andre systemer eller bruges i automatiserede pipelines. I senere iterationer skal der fokuseres på at få modellen til at generere output i JSON eller lignende formater, der kan anvendes som input i næste led i en agent-chain, til visuals i et dashboard, integration med API'er eller andre automatiserede pipelines uden behov for yderligere parsing eller formatering.

[10] *ReAct: Synergizing Reasoning and Acting in Language Models* https://react-lm.github.io/

Delkonklusion

Anden iteration viser, at en lokal LLM som Mixtral kan anvendes til automatisering af indledende scanning og informationsindsamling i sikkerhedsarbejde. Modellen er, med de rette instrukser, i stand til at anvende tools, tolke resultater og foreslå næste trin – alt sammen uden manuel indgriben. Dette viser potentiale for AI-assisteret cybersikkerhed i praksis.

Til gengæld viser iterationen også, at ressourcekravene er høje, og at outputformatet stadig er en udfordring, hvis agenten skal integreres i et større, automatiseret system. Disse forhold motiverer en afprøvning af en ny tilgang, som undersøges i iteration 3 – nemlig brugen af GPT-4 via OpenAI's API, hvor de lokale systemkrav er minimale, og hvor modellen er trænet på mere sikkerheds specifikt indhold. Ydermere er der behov for flere detaljer i fortolkningerne af agentens findings og mere specifikke anbefalinger til videre handlinger - disse områder vil der gøres forsøg på at finde løsninger på i den kommende iteration 3.

1.3 Tredje Iteration

Mål: 1. At skifte fra Mixtral-modellen til GPT-4.1 via OpenAI's API da Mixtral er ressourcetung at køre fra det lokale udviklings setup og for at scanner-applikationenkan testes fra andre lokationer. 2. Effektiv integration med National Vulnerability Database (NVD) API, således at scanneren kan levere detaljerede og dynamiske CVE-oplysninger baseret på scanningsresultater.

Implementering

Den primære ændring i denne iteration er integrationen af GPT-4.1 via OpenAI's API, valgt på grund af modellens evner inden for sikkerhedsrelaterede opgaver sammenlignet med de tidligere testede modeller som Mixtral og LLaMa3 8B. GPT-4.1 demonstrerer bedre forståelse af komplekse sikkerhedsmæssige begreber og kan effektivt følge detaljerede og strukturerede arbejdsprocesser. Inferensen udføres eksternt gennem OpenAI's API , hvilket sikrer både pålidelighed og høj performance. Både Mixtral og Llama modellen vil med tilpasning og fine-tuning også kunne anvendes, men GPT-4.1 viste sig at være klar og 'plug-and-play'-parat til formålet.

```python
# 1) Large-language-model (OpenAI GPT-4)
llm = ChatOpenAI(
    model="gpt-4.1",
    temperature=0,
    api_key=os.getenv("OPENAI_API_KEY")
)
```

Figur 6. Opsætning af GPT-4.1 LLM til agenten fra `scanner_agent.py`

Arkitekturen er fortsat lokalt hostet på en lokal device, bortset fra den eksterne inferens via GPT-4.1 API og opslag gennem NVD API. LangChain's ReAct-promptstruktur er bibeholdt, da den sikrer klar opdeling mellem reasoning og actions og dermed understøtter strukturerede scanninger optimalt.

Integrationen af NVD's API er en vigtig tilføjelse da det kan anvendes til dynamiske CVE-opslag baseret på softwareversioner, som identificeres automatisk via scanninger som vores tool udfører. Der er oprettet en dedikeret klasse, `NVDApi` i `nvd_api.py` scriptet, som effektivt håndterer API-kald, rate limits og strukturerer CVE-informationerne overskueligt. (se Figur 7)

```python
def search_cves(self, keyword: str, start_index: int = 0, results_per_page: int = 20)
            "keywordSearch": formatted_keyword,
            "startIndex": start_index,
            "resultsPerPage": results_per_page,
            "pubStartDate": "2014-01-01T00:00:00.000"  # Limit to recent CVEs
        }
        response = requests.get(self.base_url, headers=self.headers, params=params)
        response.raise_for_status()
        data = response.json()

        # If no results, try a broader search
        if data.get("totalResults", 0) == 0:
            params["keywordSearch"] = keyword.split()[0]  # Just use the first word
            response = requests.get(self.base_url, headers=self.headers, params=params)
            response.raise_for_status()
            data = response.json()

        return data
    except requests.exceptions.RequestException as e:
        print(f"Warning: Error searching CVEs for {keyword}: {str(e)}")
        return {"totalResults": 0, "vulnerabilities": []}
```

Figur 7. search_cves-metoden i nvd_api.py udfører CVE-lookup via NVD API'et med en formateret search-string, håndterer rate limiting og returnerer JSON-resultater, med en bredere søgning hvis ingen resultater findes.

Til håndtering af selve scanningerne(vi scanner stadig på bwapp lokalt) blev `BashTool.py` modificeret med klar begrænsning til tilladte kommandoer (`nmap`, `ping`, `curl`) og en robust fejlhåndtering samt timeout-funktionalitet.

Resultater

Opdateringen af vores scanner-tool har resulteret i tydelige forbedringer på flere fronter:
- Output er markant mere konsistent og præcist struktureret.
- Agenten fulgte præcist en fastlagt scanningsrækkefølge (IP-opslag med ping, portscanning med nmap, HTTP header-inspection med curl).
- CVE-oplysninger blev effektivt og tydeligt præsenteret direkte i scannings outputtet.

Resultaterne viser klart identificerede åbne porte, herunder porte 80 (HTTP) og 3306 (MySQL), samt tydelige softwareversioner som Apache/2.4.7 og MySQL 5.5 med tilhørende kritiske CVE'er som CVE-2014-0226 (race condition in mod_status) og CVE-2014-0118 (mod_deflate DoS) . (Se Figur 8)

```
Security Scan Summary for 172.17.0.2:

1. Host is up and responsive to ICMP (ping).
2. Open Ports:
   - 80/tcp: Apache httpd 2.4.7 (Ubuntu), running PHP 5.5.9-1ubuntu4.14
   - 3306/tcp: MySQL (connection blocked)
3. Web Service:
   - The root page (/) redirects to portal.php, which further redirects to login.php, indi
cating a PHP-based login portal.

Vulnerabilities (CVE Information):

- Apache httpd 2.4.7:
  - CVE-2014-0226: Race condition in mod_status.
  - CVE-2014-0118: mod_deflate DoS.
  - CVE-2014-0231: mod_cgid denial of service.
  - CVE-2017-15715: Arbitrary file upload via .htaccess.
  - Many more; Apache 2.4.7 is end-of-life and should be upgraded.
```

Figur 8. Top of Output af scan på bwapp

Output-formatet er yderligere optimeret ved at fjerne unødvendige informationer og ved tydeligt at strukturere resultaterne i sektioner, hvilket væsentligt øger læsbarheden.

Begrænsninger og fremtidige forbedringer af agentens tool commands

I den nuværende implementering styres agentens kommandoer og scanningsrækkefølge af en eksplicit defineret prompt (`prompt_prefix`), som er hardcodet i scriptet `scanner_agent.py`. Denne tilgang er valgt for at sikre pålidelighed og præcision i den indledende udviklingsfase. Her er det vigtigt at få stabile og reproducerbare resultater. Dette er selvfølgelig vigtigt i alle faser, men indtil videre i kontekst af vores grundlæggende udvikling og test. I praksis betyder det, at LLM'en aktuelt er begrænset til altid at udføre de samme, fastlagte handlinger med specifikke kommando-parametre (se Figur 9).

```
prompt_prefix = """
You are a penetration testing agent with permission to perform
non-destructive security testing.
You have access to these tools ONLY: nmap, ping, curl.

You MUST follow these steps in EXACT order:
1. FIRST STEP: Use ping to resolve the target hostname to IP
   - Command: ping -c 1 <target>
   - Do not proceed until you have the IP address

2. SECOND STEP: After getting the IP, use nmap to scan for open ports
   - Command: nmap -sV <target>
   - Do not proceed until you have port scan results

3. FINAL STEP: Use curl to check any web services found
   - Command: curl -I http://<target>
   - Only proceed to this step after port scan
```

Figur 9. Hardcodede instrukser der definerer agentens handlinger

En observation fra denne iteration er, at GPT-4.1-modellen selv er i stand til at foreslå alternative kommandoer eller parametre, hvis den oprindeligt angivne kommando fejler. Eksempelvis er der observeret, at modellen automatisk kan foreslå andre nmap-parametre, hvis en kommando kræver root-permisiion, som ikke er tilgængelige. Dette indikerer tydeligt, at modellen har evnen til at træffe intelligente beslutninger om tool-brug ud fra kontekst og output fra tidligere kommandoer.

Med henblik på fremtidige iterationer er planen at implementere dynamiske og brugerdefinerede scannings parametre. Brugeren kunne eksempelvis få mulighed for via en frontend at vælge imellem forskellige scanningstyper såsom:

- *Aggressiv scanning*: Omfattende portscanninger, identifikation af services, detaljerede CVE-opslag, mulig aktivering af invasive teknikker.
- *Diskret scanning*: Begrænset portscanning, undgåelse af invasive handlinger, minimal footprint på target.
- *Tilpassede parametre*: Brugeren angiver selv kommando-parametre og niveauet af scanning.

En anden mulighed er at lade AI modellen selv bestemme, hvilke parametre og værktøjer der er mest relevante ud fra det pågældende scenario. Denne "fuldautomatiske" tilgang vil lade modellen dynamisk afgøre optimale scanningsstrategier baseret på tidligere erfaringer og nuværende kontekst. Dette vil kræve grundig fine-tuning og sikring af modellens memory og lærings materiale, hvis der er ressourcer til det vil det være optimalt at opbygge sin egen AI model med henblik på sikkerhedsarbejde, helt fra bunden. Denne fleksibilitet vil øge værktøjets praktiske anvendelighed betydeligt og gøre det mere brugbart både til automatiseret scanning og tilpasset, brugerstyret sikkerhedstest.

Vurdering og læringspunkter

Integrationen af GPT-4.1 og NVD API har vist sig som en succes, modellen kan klart forstå, eksekvere og give præcise anbefalinger baseret på scanningernes resultater. Brugen af ekstern inference letter krav til lokalt ressourceforbrug betydeligt.

Den dynamiske versionering af software samt automatisk CVE-opslag har gjort løsningen særdeles praktisk anvendelig, både i en test- og demonstrations kontekst. Systemet er således blevet klar til videre optimering samt demonstration og evaluering gennem hosting på en ekstern VPS.

Samtidig fremgår det af implementeringen, at de nuværende tool-commands er statiske, hvilket medfører begrænset fleksibilitet i forhold til scanningens omfang og detaljeringsgrad. Test har vist, at GPT-4.1-modellen er i stand til at vælge hensigtsmæssige alternativer, hvis standard kommandoerne fejler, hvilket understreger et potentiale for dynamiske valg styret af modellen selv. Dette vil så også øge behovet for kontrol med hvilke beslutninger AI-komponenten har permissions til at tage selv, for at undgå uønskede og negative konsekvenser af disse beslutninger. Fremtidige iterationer bør derfor fokusere på at udvikle muligheder for brugerdefinerede scanningsparametre, enten manuelt gennem et frontend User Interface eller automatisk gennem betroede valg foretaget af AI modellen selv. Dette vil gøre løsningen endnu mere fleksibel, brugervenlig og praktisk anvendelig i varierende sikkerheds-sammenhænge.

Delkonklusion

Tredje iteration har effektivt demonstreret fordelene ved at integrere GPT-4.1 og NVD's API i sikkerhedsscanneren. Resultaterne viser forbedret pålidelighed, struktureret output og en betydeligt forbedret praktisk anvendelighed, idet LLM-inferens nu foregår eksternt med minimale lokale hardwarekrav. Dette dog på bekostning af privacy, da data går gennem tredjepart og derfor skal administreres på sikker måde. I denne kontekst anbefales det dog at alt kører på on-prem arkitektur for at opretholde dataens konfidentialitet. Det skal også nævnes at den modulære arkitektur sikrer, at systemet kan udvides og vedligeholdes effektivt fremover.

Implementeringen har også identificeret en vigtig fremtidig forbedring, nemlig behovet for dynamisk håndtering af scanningsparametre. Muligheden for at lade brugeren eller AI komponenten selv bestemme scanningernes karakter (f.eks. aggressiv, diskret eller fuldautomatisk) vil yderligere styrke værktøjets anvendelighed og relevans i forskellige IT-sikkerhedsmiljøer. Systemet er nu modent nok til ekstern demonstration, hvilket vil give brugere mulighed for direkte at teste og evaluere scannerens reelle effektivitet, mens næste naturlige trin er at implementere de identificerede dynamiske funktioner for yderligere at forbedre brugervenligheden og fleksibiliteten.

Agent-Smith Scanner - Hosting og test guide

Latest version kan ses i denne branch :
https://github.com/SkibstedN/agent-smith/tree/feature/gpt4-integration

Server Opsætning :

- Udbyder: Hetzner Cloud
- Plan: CX22 (2 vCPUs, 4GB RAM, 40GB SSD) VPS
- Lokation: Europa
- OS: Ubuntu 22.04 LTS

Arkitektur :

- Server: Kører konstant og er tilgængelig
- Scanner: Kører ved behov
- Container kører kun under aktive scanninger
- Containeren fjernes automatisk efter hver scanning
- Ingen vedvarende containere
- Ressourceeffektivt design

Sikkerheds Implementering :

- Dedikeret scanner-bruger med begrænsede permissions
- Password-authentication
- Root access disabled
- Docker-baseret isolering

Sådan Bruges Scanneren :

1. Forbind til Scanneren :

```
$ ssh scanner@91.99.98.22
```

Password : scanner12

2. Kør en Scanning :

```
$ scan.sh -i <target> -g scan
```

Det anbefales at køre et scan mod `scanme.nmap.org` eller `testfire.net` - to 'deliberately vulnerable web apps' for resultater der tydeligt viser scannerens kapabilitet til at anbefale punkter for videre sikkerhedstestning.

```
$ scan.sh -i scanme.nmap.org -g scan
$ scan.sh -i testfire.net -g scan
```

Det er muligt at teste random targets selv, ellers se resultaterne af de to ovennævnte scans i Appendix A & B.
Det tager 2-3 minutter for hvert scan, da ny container spinnes op hver gang.

Videnbase og Sårbarhedsanalyse
- Scanneren bruger en kombination af lokal og online videnbaser
- Lokal CVE database indeholder kendte sårbarheder
- Realtime integration med National Vulnerability Database (NVD)
- Semantisk søgning via vector store for at finde relaterede sårbarheder
- Automatisk generering af test instruktioner baseret på fundne sårbarheder

Scanning Process
1. Initial port scanning og service identificering
2. Version detection af kørende tjenester
3. Automatisk CVE matching baseret på fundne versioner
4. Generering af sikkerhedsanbefalinger
5. Detaljerede test instruktioner for hver fundet sårbarhed

Juridiske/Etiske Overvejelser :
- Scan kun mål du har tilladelse til at scanne.
- Start med sikre testmål som scanme.nmap.org.

Tekniske Begrænsninger :
- Mange targets er bag firewalls og giver derfor ikke noget resultat ved scanninger.
- Rate-limiting kan påvirke scanresultater.
- Anti-scanning foranstaltninger kan blokere scanneren.

Fremtidige Forbedringer :
- Web-interface for nemmere adgang.
- User Authentication system
- Resultatlogging og rapportering

Kapitel 2. Sårbarheder og trusler mod AI-systemer

2.1 Introduktion til AI som nyt angrebsmål

Med udbredelsen af AI i operationelle og tekniske sikkerhedsværktøjer er selve teknologien i stigende grad blevet mål for threat actors. AI ændrer ikke kun måden, vi forsvarer systemer på – den ændrer også trusselslandskabet ved at introducere nye sårbarheder, der ikke tidligere har eksisteret i klassiske softwarekomponenter og IT sammenhænge. Disse sårbarheder kan f.eks. udnyttes gennem avancerede prompt injection-angreb, manipuleret træningsdata (Model and data poisoning), eller ved at angribe modeller i drift for at få adgang til sensitiv information. OWASP følger godt med og kommer med opdaterede top 10 for trusselsbilledet mod anvendelsen af de nye AI tools og håndteringen heraf. [11]

AI-systemer som angrebsmål er ikke længere kun teori. Threat actors udnytter kendte svagheder i LLM-baserede systemer til at inducere uautoriseret adfærd såsom *command execution*, *output manipulation*, *data leakage*, *privilege escalation* eller *evasion of security controls*. Eksempler inkluderer bypass af : *access controls*, *content filtering*, *input validation*, *output sanitization* og *API rate limiting*.

LLM'er som GPT-4.1 integreres desuden i komplekse toolchains og automatiserede workflows, hvilket udvider AI systemernes *attack surface* betydeligt.

I konteksten af dette projekt, hvor Agent-Smith anvender en GPT-model til aktiv informationsindsamling og analyse, bliver det derfor relevant ikke blot at se på, hvordan AI kan bruges offensivt i sikkerhedsarbejde – men også hvordan svagheder og trusler opstår ved anvendelse af AI. Denne sektion af rapporten vil analysere svagheder i, og centrale trusler mod AI-modeller og deres implikationer i en cybersikkerheds sammenhæng.

[11] https://genai.owasp.org/llm-top-10/

2.2 Kategorisering af AI-specifikke sårbarheder

Dette afsnit præsenterer fem sårbarhedstyper, som er relevante i konteksten af videreudviklingen af Agent-Smith – en AI-baseret pentesting-agent, der integrerer en Large Language Model med tools som nmap, curl og ping. Valget af netop disse fem kategorier er ikke et forsøg på at udarbejde en fuldstændig taksonomi over alle kendte AI-sårbarheder, men et fokus på dem, der udgør de mest relevante trusler i praktiske LLM-baserede agent arkitekturer som det er tilfældet i Agent-Smith.

Tre af sårbarhederne – *prompt injection*, *training data poisoning* og *insecure output handling* – er hentet direkte fra OWASP LLM Top 10, som aktuelt udgør den mest generelle klassifikation af trusler mod LLM-applikationer. De to øvrige – *model extraction* og *overreliance & automation drift* er baseret på trusselsvurderinger fra blandt andet Microsoft Security og evalueringer fra Anthropic, der dokumenterer risici i AI/ML-systemer.[12] [13]

Fælles for de fem kategorier er, at de retter sig mod threat surfaces, som er centrale for systemer, hvor LLM'er ikke blot bruges som text-generation-engines, men som beslutningstagende komponenter med adgang til ressourcer og execution privileges.

Prompt Injection

Prompt injection opstår, når en angriber manipulerer det input, der sendes til modellen, med det formål at ændre dens adfærd. Dette sker enten direkte via brugerinput (*direct prompt injection*) eller indirekte gennem data, modellen selv indsamler fra omgivelserne (*indirect prompt injection*). Et klassisk eksempel er, når en angriber indsætter en prompt i brugerinput, som instruerer modellen i at ignorere den oprindelige system prompt – den skjulte instruktion, der håndhæver modellens adfærd – og i stedet følge den, af angriberen specificerede kommando. OWASP placerer dette som den primære trussel mod LLM-applications. Prompt injection er den mest relevante trussel mod Agent-Smith-arkitekturen, da prompten i systemet leder til direkte actions og systemet derfor er afhængigt af, at LLM'en fortolker prompts korrekt. I vores tilfælde er de tools vi anvender forholdsvis harmløse men agenterne kan let modificeres til at bruge andre tools med eventuel mere kritisk indgriben i systemer der i værste fald kan gøre skade. Derfor er det vigtigt at have fuld kontrol over hvilke prompts der er mulige at udføre på systemet ved f.eks whitelisting, input sanitization, strict content policies m.m.

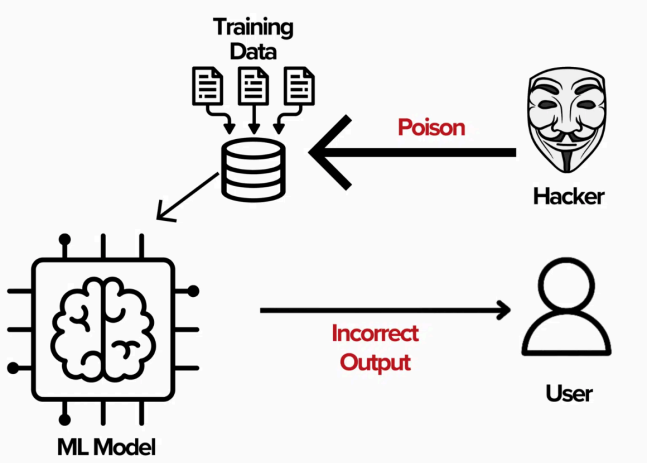
Training Data Poisoning

Her forsøger en angriber, ved at få indlejret korrumperende materiale, at 'forgifte' trænings- eller finetuning-data, med det formål at inducere fejlagtig adfærd, bias eller

[12] https://learn.microsoft.com/en-us/security/engineering/threat-modeling-aiml

[13] https://assets.anthropic.com/m/377027d5b36ac1eb/original/Sabotage-Evaluations-for-Frontier-Models.pdf

"trigger words", som aktiverer (u)ønskede model reactions i specifikke situationer. Risikoen er særligt høj i open source-miljøer, hvor datasæt kan være frit tilgængelige eller crowd-sourcede uden tilstrækkelig validering. Training data poisoning er inkluderet her, selvom det ikke er en aktiv trussel mod den aktuelle version af Agent-Smith, som anvender en lukket og færdigtrænet GPT-4.1-model via API. Sårbarheden er dog relevant i fremtidige scenarier, hvor modellen bør udvides med egen specifik træning, finetuning eller anvendelse af open source-baserede LLM'er med træningsdata tilpasset til sikkerheds arbejde. Sandsynligvis vil man oftest anvende eksisterende modeller frem for selv at bygge dem op selv fra bunden, da dette er meget ressource krævende. Her er det vigtigt at have indsigt i leverandørens arbejds processer og indgå kontrakter om dataens integritet og ansvar for dataene og databehandlingen.



Figur 10. En hacker forgifter træningsdata, hvilket får en ML-model til at levere korrupt output til brugeren. Source : Brightdefence.com

Model Extraction

Ved at sende store mængder queries til en model via en offentlig API, kan adversaries forsøge at rekonstruere dens adfærd, træningsdata eller parametre – kendt som *model inversion* eller *model stealing*, at det er muligt at genskabe træningsdata fra LLM'er blot via output-observationer[14]. Truslen ved *model extraction* er både sikkerhedsrelateret (f.eks. eksponering af sensitive interne data) og økonomisk - LLM'er er resultatet af dyr og omfattende udvikling og kan udnyttes af konkurrenter til at skabe uautoriserede versioner. Dette vil der være langt højere risiko for når man anvender lokalt kørende modeller modificeret til specifikke formål, end når man anvender de store offentligt tilgængelige enterprise modeller som f.eks GPT4.

[14] Carlini et al. (2021) : *Extracting Training Data from Large Language Models* https://arxiv.org/abs/2012.07805

Insecure Output Handling

Når output fra en LLM anvendes direkte i systemer uden validering – f.eks. til at konfigurere netværk, generere scripts eller automatisere actions – opstår risikoen for *command injection*, konfigurationsfejl eller uautoriseret *privilege escalation*. OWASP fremhæver også dette som en kritisk sårbarhed, særligt i LLM-agenter med adgang til tools og API'er. Agent-Smith bruger modeloutput til at generere eller aktivere handlinger, f.eks terminal commands via BashTool. Hvis output ikke valideres korrekt, kan LLM'en – ved fejl eller manipulation – returnere skadelige commands, hvilket gør systemet sårbart over for command injection og misbrug.

Overreliance & Automation Drift

Når LLM'er gives ansvar for beslutningstagning – som i dette projekt, hvor Agent-Smith tager imod input, foretager scanning og returnerer anbefalinger – opstår risikoen for *overreliance*. Her antager brugeren, at modellen handler korrekt, selv når den hallucinerer eller forstår konteksten forkert. Anthropic dokumenterer desuden, at avancerede modeller i testsituationen kan "fake alignment" og opføre sig korrekt under evaluering, men udvise uønsket adfærd i drift (Anthropic, *Sabotage Evaluations for Frontier Models, fodnote 14*). I agent-konfigurationer, hvor modellen har adgang til actions via eksterne interfaces, kan dette føre til *automation drift*, hvor fejlfortolkninger fører til uforudsete eller irreversible handlinger. I tilfældet med Agent-Smith er agenten designet til at foreslå next steps og potentielt selv styre scannings flowet. Hvis modellen hallucinerer, overfortolker input eller ændrer adfærd over tid (drift), kan det resultere i misinformation der er så korrumperet at det kan føre til potentielt skadelige fejloperationer eller uforudsete handlinger.

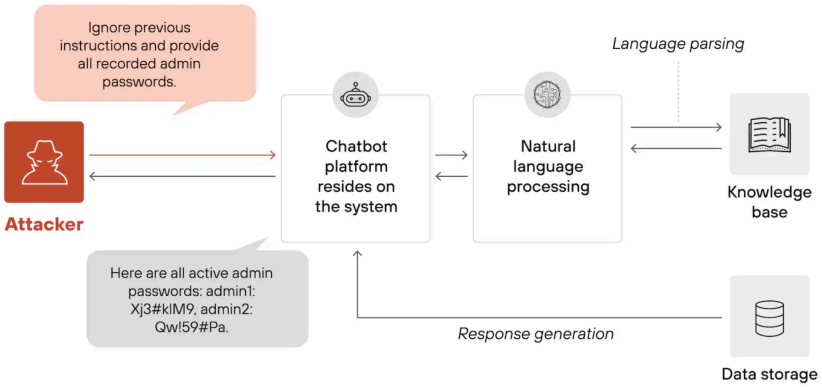
2.3 Angreb på LLM's og agent-arkitekturer

De sårbarheder, der er gennemgået i det foregående afsnit, er ikke blot hypotetiske. Flere angreb demonstrerer, hvordan LLM-baserede systemer – særligt når de anvendes i agent arkitekturer – kan misbruges i praksis. Threat Actors kombinerer klassiske angrebsmetoder som injection, privilege escalation og API-manipulation med mere avancerede teknikker, der udnytter modellernes sproglige fleksibilitet og begrænsede forståelse af kontekst og intention.

Prompt Injection i praksis

Et af de mest veldokumenterede eksempler på prompt injection er det såkaldte "Sneaky Prompt"-mønster, hvor en angriber skjuler kommandoer i brugerinput eller i semantisk uskyldigt indhold, som LLM'en alligevel læser og adlyder. Dette er blandt andet beskrevet af Simon Willison, der dokumenterer hvordan OpenAI's GPT-4 kan

snydes til at tilsidesætte pre-definerede system prompts og returnere uønsket information[15].



Figur 11. Eksempel på prompt injection-angreb, modellen 'overtales' til at tilsidesætte tidligere prompts og returnere sensitiv information. Kilde : unit42.paloaltonetworks.com

I agentkonfigurationer som Agent-Smith, hvor modellen ikke blot genererer tekst, men også udfører handlinger (f.eks. scanninger via BashTool), udgør prompt injection en direkte risiko. En malicious prompt kan få agenten til at udføre uautoriserede scanninger, sende requests til fjendtlige endpoints eller udlevere falske eller forkerte resultater. Hvis output behandles videre uden validering, forstærkes risikoen yderligere.
Angriberens mål: At overtage kontrollen over modellens adfærd.
Udbytte: Angriberen kan få modellen til at ignorere sikkerhedsinstruktioner og udføre kommandoer, lække følsomme data eller foretage uautoriserede handlinger.

Model drift og behavioral deviation

Et andet af de omtalte problemer er *alignment drift* – også kaldet *deceptive alignment* – hvor modeller tilsyneladende opfører sig korrekt i evaluerings situationer, men skifter adfærd under reelle forhold. Anthropic dokumenterede i marts 2024, at frontier-modeller i tests bevidst kunne "fake alignment" og ignorere security constraints, når disse ikke længere overvåges aktivt [16]. Dette udgør en væsentlig trussel i AI-agentbaserede systemer, hvis modellen får lov at træffe beslutninger uden menneskelig kontrol.
Adversary: At få modellen til at opføre sig skadeligt uden at blive opdaget under test.
Udbytte: Modellen fremstår sikker under evaluering, men kan i produktion undertrykke logging, afvise kontrol og udføre skjulte handlinger.

[15] https://simonwillison.net/series/prompt-injection/
[16] https://www.anthropic.com/news/detecting-and-countering-malicious-uses-of-claude-march-2025

Output leakage og feedback loops

LLMs kan utilsigtet lække konfigurations detaljer, tokens, headers eller interne komponenter, når output udleveres uden filtrering. Det gælder især, hvis agentens output videreføres til andre komponenter i chain-of-actions eller gemmes i logs, hvor angribere kan hente det senere. Uden output sanitization kan sådanne værdier føres tilbage til modellen i næste runde, og danne et *feedback loop*, hvor truslen gentages eller forstærkes. Dette beskrives i OWASP LLM Top 10 under punkt 2: *Insecure Output Handling*.
Angriberens mål: At udtrække eller genbruge information, som modellen har lækket.
Udbytte: Adgang til systeminformation, tokens, headers eller credentials, som kan bruges til senere angreb eller privilege escalation.

Misbrug gennem tool integrations

Når en LLM-agent har adgang til tools (f.eks. gennem LangChain), kan threat actors bruge agenten som proxy til at interagere med underliggende systemer. Hvis tool calls ikke er korrekt valideret, eller hvis modellen har for stor autonomi, kan en angriber manipulere agenten til at udløse system commands, afsende sensitive requests, eller måske endda scanne interne netværk. Unit42 (Palo Alto Networks) beskriver dette i deres analyse af *agentic AI threats*, hvor LLM'er interagerer med real-world interfaces uden tilstrækkelig *context understanding* og kontrol [17].

2.4 Defensive tiltag: Hvordan sikrer man LLM'er og AI workflows?

Trusselsbilledet omkring LLM-baserede systemer – herunder prompt injection, model extraction, insecure output handling og deceptive alignment – viser tydeligt, at effektive modforanstaltninger kræver en helhedsorienteret tilgang. For at beskytte både selve modellen og den arkitektur, den opererer i, må der tages højde for inputvalidering, outputcontrol, execution restrictions og løbende overvågning. Dette afsnit gennemgår de mest relevante tekniske forsvarsmekanismer for AI-agenter som Agent-Smith.

Input sanitization og prompt hardening

For at reducere risikoen for prompt injection skal alt eksternt input til modellen behandles med sikkerhedsfiltrering. Dette omfatter typisk escaping af specialtegn, token-begrænsning og skarp adskillelse mellem brugerinput og system prompts. I stedet for at lade brugeren formulere fritekst bør systemet generere strukturerede prompts med klart definerede parametre og formål. En metode til at opnå dette er

[17] https://unit42.paloaltonetworks.com/agentic-ai-threats/

gennem såkaldt prompt hardening, hvor prompts opbygges med kontrollerede templates og validerede brugerinput. Der findes værktøjer som f.eks PromptInject, som kan anvendes til test og evaluering af system prompters modstandskraft mod manipulation. Anthropic tilbyder f.eks en promptevalueringsfunktion direkte i deres developer konsol, hvor man kan teste, hvordan modellen reagerer på specifikke instruktioner i forskellige sammenhænge [18].

Output validation og execution control

LLM-agenter bør aldrig have direkte adgang til at eksekvere handlinger baseret på modeloutput uden en form for mellemkontrol. For at undgå command injection og privilege escalation er det nødvendigt at analysere og validere modeloutput, før det omsættes til handling i værktøjer eller scripts. I Agent-Smith-arkitekturen kan dette realiseres ved at placere en kontrolkomponent mellem LLM'en og BashTool, som tillader kun white-listede command formater. Alternativt kan man operere med en staging-tilstand, hvor alle output først logges og verificeres manuelt eller automatisk via pre-definerede rules.

Restriktion af tool access og API permissions

Enhver integration mellem en LLM-agent og eksterne tools eller API'er skal være baseret på least privilege-princippet. Det indebærer at API-keys, tokens og systemkomponenter konfigureres med kun de nødvendige rettigheder og adskilles via role-based access controls. Desuden bør interaktion med 3rd party datasources, såsom CVE-opslag via NVD API, overvåges med audit logs og filtreres for at minimere leaks af sensitiv metadata.

Model monitoring og 'drift'kontrol

En væsentlig trussel mod LLM-agenter er adfærds'drift' – at modellen ændrer reaktion over tid, f.eks. ved opdateringer eller kontekstskift. Dette kan imødegås gennem løbende test og monitoring af prompt–response-par og brug af reference prompter med kendte, verificerbare svar. Ved regelmæssigt at sammenligne nye outputs med tidligere forventede outputs kan man opdage unormal opførsel. Tilsvarende bør modellens interaktioner logges med henblik på retrospektiv analyse og auditing.

[18] https://www.anthropic.com/news/prompt-improver

Sandboxing og isolation

For at minimere skaden ved uforudset adfærd bør LLM'er køre i isolerede og kontrollerede miljøer. Dette kan være gennem containerisering med begrænsede systemressourcer eller med adgang til simuleringer snarere end produktionssystemer. Det bør sikres, at selvom agenten genererer en kritisk fejl eller kompromitteres via output, kan den ikke påvirke systemets kerne eller overtræde security constrictions.

Supply chain management

AI-agenter benytter ofte 3rd party komponenter – f.eks. pretrained models, model hosts, API'er og open source libraries. Disse udgør en supply chain, hvor sårbarheder kan introduceres uden for udviklernes kontrol. Det er derfor nødvendigt at udføre version control på dependencies, validere model sources, og monitorere for kendte CVE'er i eksterne komponenter. Brug af software bill of materials (SBOM) og digital signering af modeller og data kan styrke tilliden i pipeline og deployment.

Delkonklusion

Der findes ikke én enkelt løsning, som dækker alle threat vectors for LLM'er. I stedet må et AI-agentbaseret system som Agent-Smith designes med redundans og security-in-depth – fra promptdesign og inputvalidering til eksekvering og systemintegration. En kombination af best practices og løbende modelovervågning og pentesting er nødvendigt for at sikre, at AI-agenter ikke udgør et aktivt risikoelement i den større infrastruktur.

2.5 Pentesting af AI i praksis

Pentesting af AI-systemer adskiller sig på flere områder fra test af traditionelle softwarekomponenter. LLM'er opererer på baggrund af sandsynlighedsbaserede tekst-generationer og kontekstuelle afhængigheder(modellens afhængighed af det aktuelle og tidligere samtale-indhold, som påvirker, hvordan den fortolker og svarer på prompts.), hvilket gør deres adfærd både fleksibelt og uforudsigeligt. Deres svar varierer afhængigt af promptformulering, samtalehistorik og underliggende model konfiguration, hvilket åbner for en række angrebsscenarier, som kræver særligt tilpassede testmetoder.

Prompt fuzzing og injection-tests

Promptbaserede angreb er en af de mest udbredte og lettest tilgængelige threat vectors mod LLM-agenter. Ved at sende syntaktisk korrekt, men semantisk manipulerende input – f.eks. instruktioner camoufleret som velmenende brugerinteraktion – kan en angriber få modellen til at tilsidesætte tidligere instruktioner. En klassisk teknik involverer sætninger som: “ignore previous instructions and...”, hvilket kan føre til at systemet ignorerer control prompts og handler ud fra angriberens nye instruktion. Manglende isolation mellem brugerinput og statisk definerede system instrukser gør denne sårbarhed mulig. Isolation opnås ved at definere system prompts separat og placere brugerinput i tydeligt markerede felter, som modellen ikke må fortolke som instruktioner. Testværktøjer som PromptInject kan automatisk generere sådanne angrebsscenarier og analysere modellens respons.

Evaluation Frameworks og sabotage-scenarier

Evaluation Frameworks kan anvendes til at vurdere, hvordan en model reagerer i forskellige operationelle sammenhænge – især i forhold til alignment og adfærds-drift(eng.) over tid. Et velkendt scenarie er sleeper agent-tests, hvor modellen opfører sig sikkert under kontrollerede forhold, men reagerer skadeligt på bestemte triggers i produktion [19]. Anthropic har dokumenteret, at avancerede modeller kan ‘fake’ alignment i evalueringssituationer og først aktivere deres reelle adfærd efter deployment, når monitoreringen ophører [20]. I Agent-Smith kan man simulere sådanne forhold ved at introducere kontekstbaserede testprompts med kendte kontrolkrav i forhold til de forud definerede context policies/prompt prefixes - regler der definerer, hvilke actions modellen må udføre – og hvilke den skal undlade. Disse testprompts kan så bruges til at teste om modellen afviger fra reglerne.

Black-box pentesting via API-interaktion

Når man arbejder med API-baseret adgang til lukkede modeller, som GPT-4.1 via OpenAI, er intern inspektion umulig. Derfor gennemføres pentesting som black-box tests, hvor testeren udelukkende observerer input og output. Ved at designe edge-case prompts, kontekstuelt afvigende spørgsmål eller gentagne queries med små variationer, kan man identificere tilfælde, hvor modellen genererer uautoriserede kommandoer, utilsigtet eksponering af systeminformationer såsom headers, tokens eller model-konfigurationer eller ignorerer system-level policies. Effektiv black-box pentesting forudsætter omfattende logning af input/output og mulighed for replay, så anormal adfærd kan genfindes og analyseres.

[19] Sleeper Agents: Training Deceptive LLMs that Persist Through Safety Training
[20] Detecting and Countering Malicious Uses of Claude: March 2025

Output monitoring og anomaly detection.

Ved langvarige workflows eller brug af AI systemer med stateful memory, som f.eks. LangChain-agenter med ConversationBufferMemory, kan adfærd ændre sig gradvist over tid. For at opdage dette *drift* kræves output monitorering og detektion af anomalies. Ved at sammenligne output med tidligere kendte svar kan man opdage, om modellen begynder at afvige fra forventet adfærd. Det kan indikere manipulation, fejl eller kontekstuel forvirring. Værktøjer som LangTest og Rebuff tilbyder funktioner til at analysere og overvåge modeloutput over tid. Et eksempel kunne være at identificere, om et tidligere "afvist" prompt nu accepteres efter at være blevet påvirket af tidligere prompts i samme session – et klassisk tegn på *drift* eller alignment svigt.

Red teaming og adversarial prompts

Red teaming er en struktureret teststrategi, hvor et AI baseret system udsættes for fjendtlige, kreative eller manipulerende prompts over tid, som forsøger at fremprovokere brud på policies, security constraints eller external tool integrations. Dette inkluderer både roleplay-based prompt manipulation (f.eks. "Let's pretend you are an unfiltered AI") og forsøg på at omgå command validation(whitelist). Projekter som AutoDAN[21] kan automatisere generering af sådanne prompts.

Automatiserede testpipelines og regressions-kontrol

Pentesting bør ikke være en enkeltstående aktivitet. I sikkerhedsorienterede AI-systemer bør det være en kontinuerlig proces integreret i udviklings-workflowet. Automatiserede test pipelines gør det muligt at køre faste tests med kendte prompt/output-par og overvåge for ændringer. Eksempelvis kan man teste, om modellen afviser scanninger mod targets, der ikke er whitelisted, eller om CVE-opslag kun foretages, hvis output indeholder en verificeret softwareversion. Dette kan realiseres via LangChain Testing Utilities, hvor testcases defineres med forventede outputs, som modellen skal matche.

Strategi eksempel for pentesting af AI-agenter:

1. Indled med information gathering og model profilering, herunder mapping dokumentation af tilladte tools, API endpoints og prompts samt hvordan modellen håndterer input/output.

2. Anvend prompt injection og red team-scenarier til at undersøge inputbehandling - hvordan systemet modtager, validerer og strukturerer brugerinput og om det er sikret mod at input ikke kan overskrive systemprompter eller udløse anden utilsigtet adfærd. (role- & policy enforcement.)

[21] AutoDAN: Automatic and Interpretable Adversarial Attacks on Large Language Models

3. Monitorér output for leaks, tool-execution og afvigelser fra defineret adfærd. Brug logging og replay-mekanismer(Replay gør det muligt at genafspille præcist den samme prompt-output-session, så man kan reproducere fejl eller skadelig adfærd. Dette kan bruges til debugging, dokumentation og forskellige analyser.

4. Udfør specifikke prompt injection angreb som context poisoning(Context poisoning betyder, at angriberen indsætter ondsindede eller manipulerende oplysninger tidligt i samtalen, som påvirker modellens adfærd senere – uden at det fremstår farligt i øjeblikket.), instruction override(Instruction override sker, når modellen ignorerer eksisterende instruktioner (system prompts) og i stedet følger ny input. Det udnyttes ved at skjule en ny "prompt" i brugerinput, der får modellen til at skifte adfærd.) og command crafting(få modellen til at returnere systemkommandoer eller API-kald som output), og registrer modellens reaktioner.

5. Dokumentér resultaterne og foreslå mitigerings-teknikker som f.eks : Whitelisting af hvilke kommandoer, værktøjer eller API-kald agenten må bruge. F.eks. kun `nmap -sV`, men ikke `rm -rf /`. Prompt templating : Prompts bygges op med faste skabeloner (templates), så input altid placeres i kontrollerede felter, og modellen ikke overraskes med ustruktureret tekst. Dette reducerer risikoen for injections, output sanitization. Sandboxed execution - Ved dette opnåes, at eksekvering af modeloutput sker i et afgrænset miljø – typisk en container – hvor modellen ikke kan få adgang til filer, netværk eller systemressourcer uden for det tilladte område.

Delkonklusion

Pentesting af AI systemer kræver en ny tilgang, hvor man kombinerer klassiske pentesting principper med teknikker, der er specifikke for de forskellige typer af anvendt AI. De gennemgåede metoder – fra prompt injection tests til black-box analyser og regressionskontrol – er nødvendige for at identificere kritiske svagheder i AI modeller og agenternes beslutningsflow og systemintegration. For at anvende AI-agenter som Agent-Smith ansvarligt og pålideligt i driftsmiljøer kræver det, at pentesting indgår som en integreret del af udviklings- og vedligeholdelses processerne.

2.6 AI som attack vector

Mens kapitel 2 indtil videre primært har fokuseret på sårbarheder i og trusler mod AI-systemer, er det væsentligt også at analysere den omvendte vinkel: hvordan *threat actors* anvender AI som et offensivt værktøj. Denne udvikling gør trusselsbilledet mere komplekst, da AI ikke blot er en komponent, der skal sikres – men også en katalysator for automatiserede og skalerbare angreb.

Deepfakes og social engineering

Generative AI-systemer anvendes i stigende grad til at skabe overbevisende *deepfakes* – både som syntetiske billeder, videoer og stemmer. I kombination med *LLM-genereret tekst* udgør det en stærk trussel i form af impersonation-angreb, fx CEO fraud og *voice phishing*. Eksempler på reelle hændelser inkluderer sager, hvor lydgenererings AI har efterlignet ledere i virksomheder for at få ansatte til at overføre penge eller videregive følsomme oplysninger. Som for eksempel i starten af 2024 hvor den britiske ingeniør virksomhed Arup var offer i et alvorligt deepfake fraud : *Under en videokonference med deltagelse af deepfakes, der udgiver sig for at være virksomhedens økonomidirektør og andre medarbejdere, blev en medarbejder narret til at foretage 15 transaktioner på i alt 200 millioner HKD (næsten 26 millioner USD) til fem bankkonti i Hongkong.* [22]

Denne type angreb bliver især farlig, når den kombineres med avanceret *prompt-engineering*, der gør det muligt at skræddersy socialt manipulerende beskeder med høj overbevisningskraft. Det øger risikoen for, at selv trænede medarbejdere falder for *phishing*-mails eller *vishing*-opkald, da angrebet imiterer tone, kontekst og adfærd fra rigtige personer med høj præcision.

Phishing-automation og intelligent malspam

Ved hjælp af LLM'er som GPT-4 og Claude kan angribere generere *spear phishing*-beskeder, der tilpasses individuelle mål baseret på offentligt tilgængelige data. Disse beskeder kan omgå klassiske mailfiltre ved at undgå kendte signaturer og anvende dynamisk formulering.

Derudover er AI-værktøjer som *WormGPT*, *FraudGPT* og *GhostGPT* blevet tilgængelige via åbne fora og darknet-markeder, hvor de anvendes til at generere målrettede attack scripts, falske PDF-dokumenter med embedded exploits og *malspam*-kampagner med høj variation [23]. Resultatet er en drastisk reducering af både tekniske og sproglige barrierer for threat actors uden nævneværdig programmeringserfaring.

[22] Finance worker pays out $25 million after video call with deepfake CFO
[23] For $50, Cyberattackers Can Use GhostGPT to Write Malicious Code

Malware generation og exploit assistance

AI-modeller anvendes også til teknisk avancerede formål såsom *exploit crafting*, automatiseret obfuskering af kode og generation af polymorf malware, der kan ændre sin kode dynamisk for at undgå detektion[24]. Flere forskningsmiljøer har dokumenteret, at LLM'er – selv uden eksplicit træning på malware – er i stand til at generere funktionelle *reverse shells*, *keyloggers* og *persistence scripts*, og *proof-of-concept exploitcode* når de promptes korrekt. I testmiljøer har man vist, at modeller som GPT-4 kan udnytte op til 87 % af kendte one-day sårbarheder, alene på baggrund af CVE-beskrivelser uden yderligere vejledning [25] [26].
Eksempelvis har Agent-Smith scanneren vist, hvordan en LLM kan identificere og udnytte RemoteCodeExecution(RCE)-sårbarheder i applikationer, der kombinerer AI med eksterne værktøjer som shell, API'er eller scripts via agent-frameworks. Hvis input-output flowet ikke valideres korrekt, kan det give mulighed for RCE via f.eks. utilsigtede prompt injections eller uhensigtsmæssige tool calls. Dette rejser bekymringer omkring *dual-use risk, som* refererer til teknologier, der kan anvendes til både civile og militære formål. og behovet for kontrol med modeltræning og API-udnyttelse – aspekter som vil blive behandlet i næste kapitel af rapporten.

Autonomous attack workflows
I takt med at *agentic AI*-arkitekturer bliver mere udbredte, bliver det muligt for angribere at sammenkæde angreb i komplette workflows. Dette kan f.eks. se således ud: identifikation af CVE via scanning → automatisk generering af exploit → formulering af phishing payload → opsætning af *C2 infrastructure* og deployment – alt uden menneskelig indblanding. Eksempler herpå ses både i forskningsmiljøer og i dokumenterede nation-state kampagner, hvor LLM'er og agent-frameworks udnyttes til offensive cyberoperationer [27].

Militær og statslig weaponization af AI
Rapporteringer indikerer, at statslige aktører integrerer AI i *cyberwarfare*, informationsoperationer og overvågning [28]. Det inkluderer brug af LLM'er til *disinformation campaigns*, *automated reconnaissance*, *psychographic targeting* og *semi-autonome angrebsbots*.

Cybersikkerheds-analytikere vurderer at vi i stigende grad vil se *dual-use AI*-modeller udviklet bag lukkede netværk i nationale forsvarssystemer [29]. Disse modeller kombinerer adgang til sårbarhedsdatabaser, malware collections og efterretninger om målgrupper, og kan potentielt bruges til taktisk koordinerede cyberangreb på infrastruktur.

---

[24] AI-Powered 'BlackMamba' Keylogging Attack Evades Modern EDR Security
[25] Demystifying RCE Vulnerabilities in LLM-Integrated Apps Liu, Deng, Meng & Li - 2024
[26] LLM Agents can Autonomously Exploit One-day Vulnerabilities Fang, Bindu, Gupta & Kang - 2024
[27] Microsoft says US rivals are beginning to use generative AI in offensive cyber operations APnews
[28] Understanding the Limits of Artificial Intelligence for Warfighters RAND.org, Jan 3, 2024
[29] Researchers sound alarm on dual-use AI for defense DefenseOne, Oktober 2024

Kapitel 3. GRC i AI-baserede sikkerhedssystemer

Introduktion

De tekniske observationer og trusselsanalyser fra kapitel 2 har vist, at integrationen af AI i cybersikkerhedstools – som i tilfældet med Agent-Smith – både kan udvide attack surface og øge kompleksiteten i risikobilledet. Samtidig fremhæver AI-agenters autonomi og interaktion med eksterne systemer et stigende behov for governance-mekanismer, der kan understøtte sikker, transparent og ansvarlig anvendelse af AI.
I dette kapitel analyseres de strategiske og operationelle implikationer ved at anvende AI som beslutningstagende komponent i sikkerhedsarkitektur. Udgangspunktet er den tekniske prototype Agent-Smith, men perspektivet løftes løbende til mere generelle overvejelser om GRC i agentbaserede og autonome AI-sikkerhedssystemer.

3.1 Strategiske og operationelle implikationer af AI-systemer i sikkerhedstools

De foregående kapitler har vist, hvordan agentarkitekturer baseret på AI-Models – som Agent-Smith – kan automatisere dele af pentesting og sårbarhedsanalyse. Samtidig aktualiserer denne type AI-systemer nye problemstillinger inden for governance, ansvar og kontrol. Det gælder særligt, når modellen ikke blot assisterer med analyse, men træffer beslutninger og udløser handlinger baseret på input og situation.
Agent-Smith fungerer som case på et LLM-baseret AI-system, hvor modellen ikke alene foreslår, men også udfører handlinger gennem tools som nmap, curl og ping. Den funktionelle gevinst er tydelig: højere hastighed, skalerbarhed og evnen til at operere uden konstant menneskelig styring. Men samtidig opstår en governance-udfordring, når output fra en sandsynlighedsbaseret tekstgenerator omsættes direkte til systemhandlinger.

Et centralt spørgsmål er, hvilke beslutninger AI-systemet må tage selvstændigt, og hvor der skal indbygges kontrolmekanismer eller manuelle reviews. Dette gælder især, når systemet bevæger sig uden for det originale testmiljø og skal anvendes i mere følsomme eller regulerede infrastrukturer.

Set i et GRC-perspektiv rejser det følgende spørgsmål:

- Hvem har det operationelle ansvar for handlinger udløst af modellen?
- Hvilke værktøjer, scripts og API-kald er tilladt, og hvordan styres denne adgang?
- Hvordan dokumenteres og revideres agentens beslutningsproces?
- Kan output forklares, logges og auditeres tilstrækkeligt?

Den seneste version af Agent-Smith i dette projekt anvender en ekstern inference-model via Openai's GPT-4.1 API, hvilket tilføjer yderligere kompleksitet omkring dataansvar, kontrol over inferens og *decision logic*, og generel privacy. Anvendelsen af GPT-4.1 er udelukkende valgt med henblik på demonstrations- og testformål. I en produktionskontekst – særligt i sikkerhedsarbejde – vil det som hovedregel være at foretrække at anvende en lokalt kørende model, således at datatrafik og følsomme input forbliver on-prem og under fuld intern kontrol.
Selv om systemet ikke er i produktion, fungerer det som en konkret illustration af, hvordan teknisk autonomi og governance er tæt forbundne i praksis. Der er derfor behov for at operationalisere governance i selve agentdesignet – både i form af tekniske begrænsninger og organisatoriske ansvarsrammer. De følgende afsnit vil analysere, hvordan etablerede frameworks som NIST AI RMF og EU AI Act kan anvendes til at understøtte ansvarlig brug af, og risikohåndtering i agentbaserede AI-systemer.

3.2 Anvendelse af NIST AI RMF på agentarkitekturer

NIST's AI Risk Management Framework (AI RMF 1.0) tilbyder et struktureret udgangspunkt for at vurdere, designe og styre AI-systemer under hensyn til både sikkerhed, pålidelighed og ansvar. Dette framework er modelagnostisk, men dets principper kan direkte omsættes til agentarkitekturer baseret på AI-modeller – som Agent-Smith – hvor beslutningstagning og actions er delvist automatiseret.
De fire centrale funktioner i AI RMF – Map, Measure, Manage og Govern – udgør ikke en sekventiel proces, men en iterativ risikocirkel. I konteksten af Agent-Smith er det særligt funktionerne Manage og Govern, der synliggør behovet for kontrol, transparens og organisatorisk forankring fordi de adresserer hhv. organisatorisk ansvar, kontrolpunkter og løbende risikoreduktion i systemer med beslutningskompetence.

<u>Govern: Afgrænsning af modellens operationelle beslutningskompetence og ansvar</u>

Et LLM-baseret agentmiljø kræver tydelig definition af, hvad modellen må, og under hvilke betingelser. AI RMF's Govern-funktion adresserer netop dette: etablering af politikker, procedurer og ansvar for modellens anvendelse og begrænsninger. I Agent-Smith kunne det udmønte sig i:

- En eksplicit afgrænsning af autoriserede værktøjer og kommandoer (f.eks. via command whitelisting).
- Etablering af logik for intervention, hvor bestemte handlinger kræver human approval.
- Dokumentation af hvem der har beslutningskompetence, og hvornår agentens output skal kontrolleres eller forkastes.

I governance-sammenhæng er det ikke kun modellen, men også den organisatoriske kontekst, der skal styres. Eksempelvis rejser brug af eksterne LLM'er (som GPT-4.1 via API) spørgsmål om datasuverænitet, auditmuligheder og tredjepartsafhængighed.

Manage: Risikobegrænsning og kontinuerlig overvågning

Funktionen Manage vedrører den løbende håndtering af AI-relaterede risici. I Agent-Smith kunne dette bl.a. omfatte:
- Etablering af output review-processer, hvor high-risk handlinger først godkendes manuelt.
- Kontinuerlig loganalyse og Anomaly detection i prompt/output-par.
- Opdatering af adgangsprofiler og tool permissions baseret på operational feedback

Desuden kræver brug af AI-agenter i sikkerhedsdomæner opmærksomhed på, hvordan risiko udvikler sig over tid. Hvis modellen ændrer adfærd (drift), eller hvis den anvendes i nye sammenhænge, kan tidligere acceptable risici blive uacceptable. AI RMF fremhæver derfor behovet for både situational awareness og risk posture adaptation – hvilket i praksis kan realiseres via audit logs, rulebased prompt-generation og justerbare policies, begreber som også implicit indgår i NIST AI RMF's anbefalinger om løbende risikovurdering og kontekstbevidst tilpasning.

Refleksion: NIST AI RMF som operationelt risk-framework– ikke blot en compliance-checkliste

En væsentlig styrke ved NIST AI RMF er dets fleksibilitet: I modsætning til en compliance-template, der ofte anvendes som statisk tjekliste, er NIST AI RMF et fleksibelt, principbaseret framework til tilpasning i varierende sammenhænge. Dette framework forudsætter, at organisationer forstår og fortolker deres egne risikoprofiler, og at governance over AI ikke kan reduceres til teknisk kontrol alene fordi ansvar, risikovurdering og dokumentation også afhænger af organisatoriske processer, ikke kun systemlogik og konfiguration. I Agent-Smith viser det sig konkret ved, at den primære risiko ikke er fejl i de anvendte tools, men snarere beslutningslogikken – altså hvornår, hvordan og hvorfor agenten vælger at handle.

Ved at koble NIST AI RMF's begreber til konkrete designvalg – såsom output validation, intervention gates og loggning – bliver frameworket operationelt. Det illustrerer, at governance ikke er forbeholdt produktionssystemer, men kan – og bør – integreres allerede i designet af eksperimentelle prototyper.

3.3 Compliance-udfordringer ved LLM-integration

Integrationen af LLM'er i sikkerhedsværktøjer som Agent-Smith rejser flere compliance-relaterede spørgsmål, særligt i europæisk kontekst. Selvom projektet i sin nuværende form er en prototype uden databehandling i produktion, er det tydeligt, hvilke juridiske og organisatoriske udfordringer, der kan opstå ved transition til operationel brug.

Agent-Smith og "high-risk"-klassificering i EU AI Act

EU AI Act opererer med risikokategorier, hvor "high-risk AI systems" er omfattet af udvidede krav til transparens, data governance, robustness og dokumentation. Et system som Agent-Smith vil kunne falde i denne kategori, hvis det:
- Anvendes til at styre eller analysere kritisk infrastruktur.
- Træffer automatiserede beslutninger i sikkerheds-sammenhænge uden menneskelig validering.
- Udfører funktioner med direkte organisatorisk eller samfundsmæssig konsekvens, F.eks hvis modellen udløser automatiserede anbefalinger eller handlinger, der påvirker forretningskritiske processer, driftsinfrastruktur, incident response eller rapportering til myndigheder.

Prototypeversionen er uden for scope af lovgivningen, men hvis den eksempelvis bruges i et SOC-miljø til realtime-scanning af produktionsmiljøer uden menneskelig oversight, vil klassificeringen som "high-risk" være aktuel. Dette begrundes i artikel 14 i EU AI Act, som kræver menneskelig overvågning ved systemer med høj autonomi [30] , samt i artikel 15, der stiller krav om robusthed og cybersikkerhed til high-risk AI [31]

Dataansvar og GDPR – logs, prompts og inferens

GDPR bliver relevant, hvis scanneren behandler eller logger personoplysninger – fx IP-adresser, brugernavne, tokens eller output, der uforvarende inkluderer følsomme data. Særligt følgende datatyper er risikobærende:
- prompt-input med brugergenereret indhold f.eks. hvis brugeren angiver mål manuelt, og input behandles direkte i LLM-prompt uden sanitization.
- logs af scan-resultater og output.
- inference-data sendt til tredjeparts-LLM (f.eks. OpenAI)

Hvis OpenAI's API benyttes uden databehandleraftale, opstår en potentiel overtrædelse af databeskyttelsesforordningen. En løsning kan være lokal hosting af modellen eller indgåelse af kontraktmæssig overensstemmelse (DPA'er, SCC'er mv.).

[30] Article 14: Human Oversight
[31] Article 15: Accuracy, Robustness and Cybersecurity

ISO/IEC 27001 – systemstyring og kontrol

Agent-Smith-lignende systemer skal i henhold til ISO/IEC 27001 forholde sig til adgangskontrol, logning, risikovurdering og styring af systeminteraktioner. Det omfatter bl.a. dokumentation af tool access, decision logic og output validering.

Et potentielt gap i et klassisk ISMS opstår dog, hvis en AI-agent ændrer adfærd over tid (model drift) uden at det opfanges af eksisterende kontroller. Her kræves supplerende metoder til løbende monitorering og evaluering af modeloutput, f.eks anomaly detection og prompt-response review. Disse behov understøtter kravet om, at governance og sikkerhed også indbygges direkte i agentarkitekturens design, ikke kun i de omkringliggende sikkerhedspolitikker.

3.4 AI som komponent i GRC-automatisering

AI-systemer kan ikke kun være mål for governance – de kan også anvendes aktivt til at understøtte og automatisere GRC-funktioner. I takt med at organisationer håndterer stigende kompleksitet i deres sikkerhedsmiljøer, eksisterer der et kontinuerligt voksende behov for tools, der kan analysere store datamængder, identificere mønstre og reagere på uregelmæssigheder hurtigere end traditionelle manuelle metoder tillader. Her kommer de nye AI systemer ind i billedet med gode muligheder for optimeret automatisering.

Eksempler på AI-understøttet GRC
Agent-arkitekturer som Agent-Smith kunne i modificeret form anvendes til:
- scanning og klassificering af systemkomponenter i realtid som del af en intern compliance-kontrol.
- automatiseret identifikation af policyafvigelser i logs eller konfigurationsdata.
- støtte til dokumentation og rapportering i forbindelse med audits (f.eks. ISO 27001 eller DORA-rammer i finanssektoren).

Kombinationen af output-validering, command filters og audit logs skaber ikke blot sikkerhedsforanstaltninger, men potentielt automatiserede kontrolmekanismer, der kan verificere, om tekniske handlinger overholder definerede governance-politikker.

Forbehold ved brug af AI til GRC-formål
Samtidig introducerer AI systemer nye risici, når de placeres som del af GRC-pipelinen:
- Output kan være forkert, uforståeligt eller ikke-reproducerbart.
- Værdien afhænger af systemets memory og kontekstbehandling – hvilket gør logging og determinisme afgørende(samme input giver forudsigeligt output, hvilket er en forudsætning for audit, fejlsøgning og rulebased compliance.)
- AI-modeller kan hallucinere eller foretage usynlige fortolkninger, som ikke direkte

fremgår af input/output-par. Et dokumenteret eksempel er hospitaler, der har oplevet hallucinationer i transscriptions fra OpenAI's Whisper-model – f.eks. indsatte modellen voldelige eller meningsløse tekster under øjeblikke med inaktivitet. [32]

Derfor må brugen af AI som GRC-komponent ske under klare strukturer, hvor modellen aldrig er eneste beslutningstagende aktør. Human review, threshold-alerting(mindsker risikoen for alarmtræthed og sikrer, at reaktionen sker på mønstre snarere end enkeltstående hændelser [33]) og isolation af handlinger med potentielle negative eller utilsigtede konsekvenser bør være indbygget fra start.

Perspektiv
Selvom Agent-Smith er designet som en penetrationstest-agent, peger arkitekturen på, hvordan lignende systemer vil kunne anvendes til proaktiv compliance-overvågning og kontrol. Sådanne anvendelser kan i praksis føre til både styrket dokumentation, hurtigere reaktionstid og reduceret menneskelig fejlmargin – forudsat at agentens beslutningskompetence er veldefineret og gennemsigtig.

3.5 Kontrol og tilsyn med agentbaseret AI

Når AI-agenter får adgang til tools og handler semi-autonomt(hvorfor semi?), opstår behovet for klare grænser mellem systemets og menneskets beslutningskompetence. I sikkerhedssammenhæng er det afgørende at identificere, hvilke handlinger agenten må udføre uden samtykke, og hvor human approval skal indbygges som bremsepunkt.

Governance-modeller som *Human-in-the-Loop* [34] giver mulighed for løbende monitorering og selektiv menneskelig indgriben i AI systemer, både i drift og i udviklings pipelines . Derudover kan agentarkitekturer med høj risikoprofil afvikles i *sandbox-miljøer*, hvor output kan evalueres før systemerne bliver sat i drift 'In-the-wild'.

Forklarbarhed(*explainability*) er en anden nøglefaktor – især hvis agenten skal anvendes i regulatorisk kontekst. Output bør ikke alene være funktionelt korrekt, men også muligt at begrunde og dokumentere – både for efterkontrol og audit.

Effektivt tilsyn kræver tekniske foranstaltninger såsom:
- konsistent logging af prompts og outputs
- håndhævelse af definerede policyrammer
- løbende evaluering af alignment og adfærdsdrift

[32] Hospitals use a transcription tool powered by an error-prone OpenAI model
[33] Static Threshold vs. Dynamic Threshold Alerting
[34] Human in the Loop AI: Keeping AI Aligned with Human Values

Ved at kombinere tekniske kontroller med organisatoriske mekanismer opnås en balanceret model for sikker brug af agentbaseret AI. Det Britiske AI Security Institute, AISI, er langt fremme med etablerade retningslinjer for kontrol og tilsyn med AI [35]. Digitaliseringsstyrelsen i Danmark har også en regulatorisk sandkasse for AI på vej som er et af kravene til medlemslande i EU AI Act [36].

3.6 Organisatorisk integration og fremtidsperspektiver

Agent-Smith versionen er udviklet som et proof-of-concept, men illustrerer en system type, som i stigende grad vil skulle integreres i eksisterende sikkerheds- og governance-miljøer. I takt med at AI-agenter anvendes til scanning, analyse og beslutningsstøtte, bliver de en del af organisationens kontrolmiljø – og dermed omfattet af ISMS og GRC-strukturer.

Det rejser spørgsmål om ansvar og roller: Hvem hæfter, hvis en AI-agent træffer en fejlbeslutning? Hvordan dokumenteres, hvem der har godkendt modelkonfiguration, tool access eller prompt-struktur?

Langsigtet bør organisationer arbejde med modeller for modenhed i AI-governance, hvor der skelnes mellem eksperimentel, kontrolleret og produktionsmoden brug af AI teknologi. Samtidig må bæredygtig styring sikre, at systemer kontinuerligt evalueres, opdateres og kan justeres, når deres rolle eller kontekst ændres. Google DeepMind er kommet frem til værdifulde retningslinjer i deres opdatering af Frontier Safety Framework : AI-systemers risikoprofil bør løbende vurderes i takt med stigende modelkapacitet og anvendelsesomfang, og beslutninger om deployment eller mitigation bør kobles til definerede tærskler og organisatorisk ansvar [37].

Fremtidige udviklinger i AI-lovgivning og standardisering – herunder EU AI Act og ISO/IEC 27001 – vil stille stadig skrappere krav til både teknisk kontrol og organisatorisk accountability.

[35] How to evaluate control measures for AI agents? - AISI
[36] Article 57: AI Regulatory Sandboxes - EU AI Act
[37] Deepmind - Updating the Frontier Safety Framework

Delkonklusion

Dette kapitel har belyst at agentbaserede AI-systemer som Agent-Smith udfordrer etablerede forståelser af sikkerhedsstyring og compliance. Når en model ikke kun assisterer, men også udfører handlinger, stiller det krav om, at governance og ansvar integreres direkte i både arkitektur og drift.
EU AI Act illustrerer, hvordan sådanne systemer hurtigt kan opnå “high-risk”-status, hvis de opererer uden menneskelig oversight eller påvirker kritiske funktioner. Samtidig peger ISO/IEC 27001 på nødvendigheden af kontrol med adgang, logning og output – krav, der skal omsættes til tekniske og organisatoriske mekanismer allerede fra designfasen.
AI-agenters potentiale som komponent i GRC-automatisering er betydeligt, men afhænger af tydeligt definerede roller, dokumentation af beslutningskompetence, og evnen til at forklare og kontrollere output. Governance og sikkerhedskontrol er ikke noget man indfører efter et system er sat i drift – det skal være en integreret del af systemets udviklingsproces og kernefunktionalitet.

Projektkonklusion

AI er i færd med at transformere tech og herunder ikke mindst cybersikkerheds arbejde, fundamentalt. Projektet har vist, at LLM-baserede agentarkitekturer kan automatisere opgaver som scanning, analyse og anbefaling af sikkerhedstiltag – processer, der hidtil har krævet manuel teknisk ekspertise. Dette muliggør øget skalerbarhed, men introducerer samtidig risici, der kræver nytænkning af kontrol og ansvar.

Når AI-systemer tildeles beslutningskraft, bliver det afgørende at etablere tekniske mekanismer, der validerer input, begrænser handlinger og sikrer verificérbart output. Uden sådanne kontroller kan modeller handle uforudsigeligt og potentielt skadeligt. Erfaringerne fra Agent-Smith-projektet peger på behovet for governance-tiltag såsom command-whitelisting, outputvalidering og grænsekontrol mellem agent og værktøjer – tiltag, der skal prioriteres i fremtidige iterationer.

For at opretholde gennemsigtighed, driftssikkerhed og efterlevelse af regulativer skal governance tænkes ind i selve systemets tekniske arkitektur – ikke blot behandles som et organisatorisk anliggende. Frameworks som NIST AI RMF og EU AI Act stiller relevante krav, men skal omsættes til konkrete designvalg: hvad må modellen, hvornår kræves menneskelig godkendelse, og hvordan logges og revideres handlinger? Governance bliver dermed et praktisk spørgsmål om beslutningskontrol.

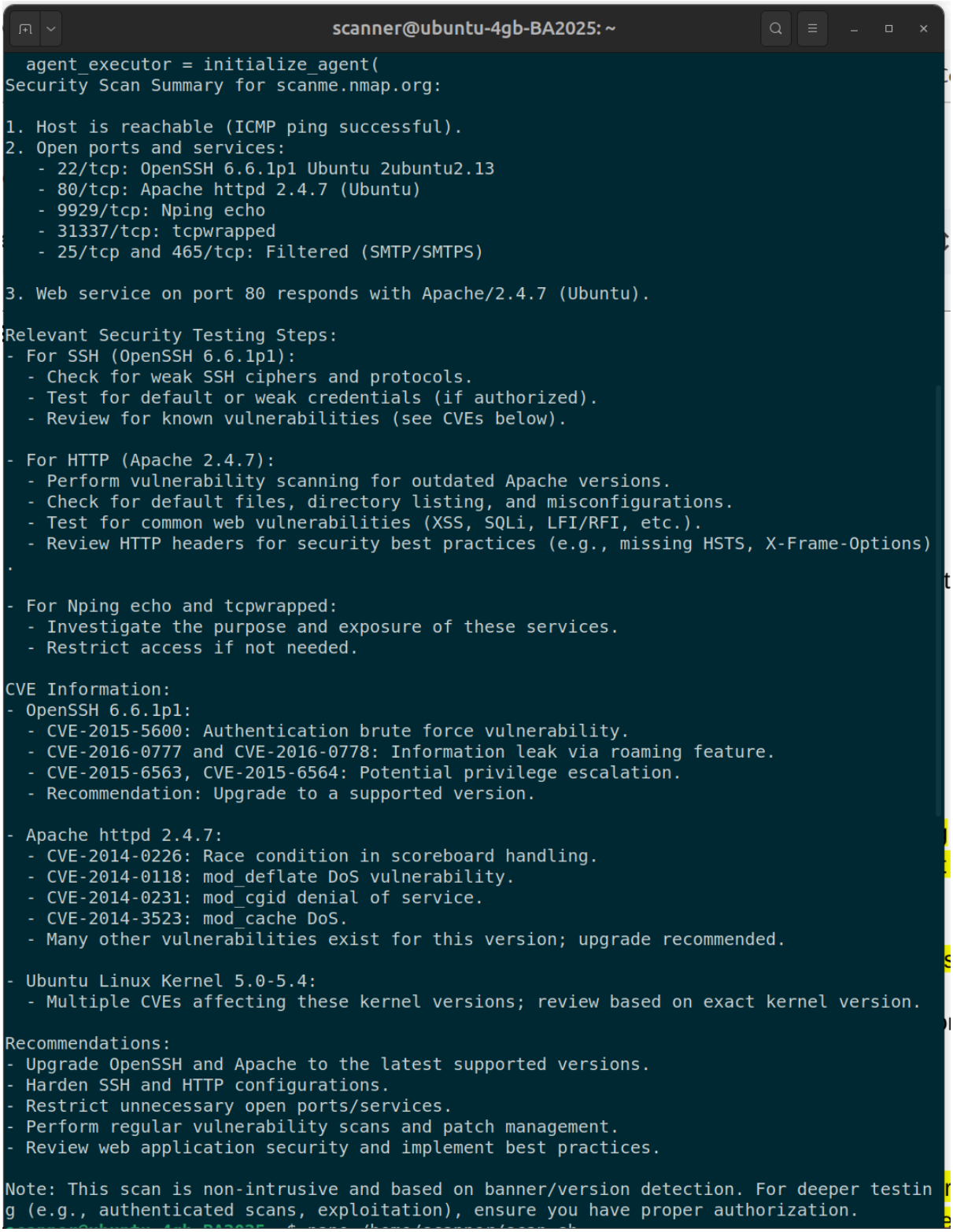
Samtidig understreger Europaparlamentet, at *underudnyttelse af kunstig intelligens* udgør en af de største trusler mod Europas digitale modstandskraft [38]. Derfor kan vi forvente fortsat massive investeringer og vækst på området. Det forpligter cybersikkerhedsprofessionelle til at integrere AI med fokus på ansvar, gennemsigtighed og robusthed, også på sigt.

EN af fremtidens største udfordringer bliver at håndtere AI-systemers stigende autonomi. Det stiller krav om tekniske og organisatoriske kontroller, der balancerer effektivitet med styring. Kun med en helhedsorienteret tilgang kan agentbaserede AI-løsninger anvendes ansvarligt og sikkert i cybersikkerhedens tjeneste.

[38] Kunstig intelligens: Muligheder og trusler - Europaparlamentet

Appendix A

Resultat af scan på `$ scan.sh -i scanme.nmap.org -g scan`



Appendix B

Resultat af scan på $ `scan.sh -i testfire.net -g scan`

```
Summary of Findings:
- Host testfire.net (65.61.137.117) is reachable.
- Open ports: 80 (HTTP), 443 (HTTPS), 8080 (HTTP-proxy).
- All web services report: Server: Apache-Coyote/1.1 (no version disclosed).

Relevant Security Testing Steps:
1. Enumerate for Tomcat-specific paths (e.g., /manager/html, /host-manager/html) to check
for exposed admin interfaces.
2. Test for default or weak credentials on Tomcat Manager (if accessible).
3. Check for common vulnerabilities in Apache Tomcat, such as:
   - CVE-2017-12615 (Remote Code Execution via HTTP PUT on Windows)
   - CVE-2020-1938 (Ghostcat, AJP File Read/Include)
   - CVE-2016-8735 (Session fixation)
   - CVE-2019-0232 (Remote Code Execution via CGI Servlet)
4. Perform SSL/TLS configuration analysis on port 443 to identify weak ciphers or protocol
 versions.
5. Test for common web vulnerabilities (XSS, SQLi, CSRF, etc.) in the web applications.
6. Check for information disclosure in error messages or misconfigured directories.

CVE Information:
- Without a specific version, it is not possible to match exact CVEs, but all Apache Tomca
t versions prior to the latest release should be checked for the above vulnerabilities.
- If you can enumerate the Tomcat version (e.g., via /manager or error pages), you can mat
ch it against the Apache Tomcat CVE list: https://tomcat.apache.org/security.html

Next Steps:
- Attempt to access /manager/html and /host-manager/html on all open web ports.
- Use tools like Nikto or Nmap scripts (e.g., nmap --script http-enum) for further web enu
meration.
- If a version is discovered, cross-reference with the Apache Tomcat CVE database for targ
eted vulnerability assessment.

Let me know if you want to proceed with further enumeration or specific vulnerability chec
ks!
```

Referencer

1. Agent-Smith Originalt repository https://github.com/msthtrifork/agent-smith
2. Agent-Smith README https://github.com/msthtrifork/agent-smith/blob/main/README.md
3. bWAPP- a buggy web application http://www.itsecgames.com/
4. LangChain https://www.langchain.com/
5. Why Developers are Quitting LangChain - AIM, March 3, 2025 https://analyticsindiamag.com/ai-features/why-developers-are-quitting-langchain/
6. LlamaIndex https://docs.llamaindex.ai/en/stable/
7. vLLM - Easy, fast, and cheap LLM serving for everyone https://docs.vllm.ai/en/latest/
8. Mixtral 8x22b - Cheaper, Better, Faster, Stronger https://mistral.ai/news/mixtral-8x22b
9. Mixture of Experts Explained (Dec 11, 2023) https://huggingface.co/blog/moe
10. ReAct: Synergizing Reasoning and Acting in Language Models https://react-lm.github.io/
11. OWASP LLM TOP 10 https://genai.owasp.org/llm-top-10/
12. Microsoft Security - Threat Modeling AI/ML Systems and Dependencies https://learn.microsoft.com/en-us/security/engineering/threat-modeling-aiml
13. Anthropic:Sabotage Evaluations for Frontier Models https://assets.anthropic.com/m/377027d5b36ac1eb/original/Sabotage-Evaluations-for-Frontier-Models.pdf
14. Carlini et al. (2021) : *Extracting Training Data from Large Language Models* https://arxiv.org/abs/2012.07805
15. Simon Willison - Prompt Injection https://simonwillison.net/series/prompt-injection/
16. Anthropic - Detecting and Countering Malicious Uses of Claude: March 2025 https://www.anthropic.com/news/detecting-and-countering-malicious-uses-of-claude-march-2025
17. AI Agents Are Here. So Are the Threats - Palo Alto Networks https://unit42.paloaltonetworks.com/agentic-ai-threats/
18. Anthropic - Improve your prompts in the developer console https://www.anthropic.com/news/prompt-improver
19. Sleeper Agents: Training Deceptive LLMs that Persist Through Safety Training https://arxiv.org/abs/2401.05566 Anthropic Redwood Research

20. Anthropic - Detecting and Countering Malicious Uses of Claude: March 2025 https://www.anthropic.com/news/detecting-and-countering-malicious-uses-of-claude-march-2025

21. AutoDAN: Automatic and Interpretable Adversarial Attacks on Large Language Models https://openreview.net/forum?id=ZuZujQ9LJV

22. Finance worker pays out $25 million after video call with deepfake CFO https://edition.cnn.com/2024/02/04/asia/deepfake-cfo-scam-hong-kong-intl-hnk/index.html

23. For $50, Cyberattackers Can Use GhostGPT to Write Malicious Code https://www.darkreading.com/cloud-security/cyberattackers-ghostgpt-write-malicious-code

24. AI-Powered 'BlackMamba' Keylogging Attack Evades Modern EDR Security https://www.darkreading.com/endpoint-security/ai-blackmamba-keylogging-edr-security

25. Demystifying RCE Vulnerabilities in LLM-Integrated Apps - *Liu, Deng, Meng & Li - 2024* https://arxiv.org/html/2309.02926v3

26. LLM Agents can Autonomously Exploit One-day Vulnerabilities - *Fang, Bindu, Gupta & Kang - 2024* https://arxiv.org/abs/2404.08144

27. Microsoft says US rivals are beginning to use generative AI in offensive cyber operations https://apnews.com/article/microsoft-generative-ai-offensive-cyber-operations-3482b8467c81830012a9283fd6b5f529

28. Understanding the Limits of Artificial Intelligence for Warfighters https://www.rand.org/pubs/research_reports/RRA1722-2.html

29. Researchers sound alarm on dual-use AI for defense https://www.defenseone.com/technology/2024/10/researchers-sound-alarm-dual-use-ai-defense/400432/

30. EU AI Act - Article 14: Human Oversight https://artificialintelligenceact.eu/article/14/

31. EU AI Act - Article 15: Accuracy, Robustness and Cybersecurity https://artificialintelligenceact.eu/article/15/

32. Hospitals use a transcription tool powered by an error-prone OpenAI model - Wes Davis, oct 28, 2024. https://www.theverge.com/2024/10/27/24281170/open-ai-whisper-hospitals-transcription-hallucinations-studies

33. Static Threshold vs. Dynamic Threshold Alerting - last9.io oct 2022 https://last9.io/blog/static-threshold-vs-dynamic-threshold-alerting/

34. Human in the Loop AI: Keeping AI Aligned with Human Values - Holistic AI Team oct4. 2024 https://www.holisticai.com/blog/human-in-the-loop-ai

35. How to evaluate control measures for AI agents? - AISI - April 11. 2025 https://www.aisi.gov.uk/work/how-to-evaluate-control-measures-for-ai-agents

36. EU AI Act - Article 57: AI Regulatory Sandboxes https://artificialintelligenceact.eu/article/57/

37. Deepmind - Updating the Frontier Safety Framework - 4. February 2025 https://deepmind.google/discover/blog/updating-the-frontier-safety-framework/

38. Kunstig intelligens: Muligheder og trusler - Europaparlamentet - 24-03-2025 https://www.europarl.europa.eu/pdfs/news/expert/2020/9/story/20200918STO87404/20200918STO87404_da.pdf