Black Hills Information Security

AI-LLM Red Teaming

OWASP Top 10 for LLM Applications

Brian Fehrman & Derek Banks

AntiSyphon Training

What We'll Cover

1. LLM Security Introduction
2. OWASP Top 10 for LLM Applications
3. Red team methodology & workflow
4. Defensive frameworks, best practices, tools

Workshop: Hacking AI-LLM Applications

Course Authored by Brian Fehrman and Derek Banks.

This workshop starts with AI-LLM application fundamentals, moving to a reference architecture based on Open WebUI, and then discusses related threats and vulnerabilities.

Live Training **$25.00**

Course Length: 4 Hours

Includes a Certificate of Completion

ENROLL NOW

Next scheduled date: March 6th, 2026 @ 12:00 PM EST

Attacking, Defending, and Leveraging AI-LLM Systems

Course Authored by Brian Fehrman and Derek Banks.

Attacking, Defending, and Leveraging AI/LLM Systems is a 16-hour, hands-on training designed for cybersecurity professionals, red teamers, and defenders who want to master how modern AI systems work, and how they can be exploited, secured, and applied in real-world operations.

Live Training **$575.00**

Course Length: 16 Hours

Includes a Certificate of Completion

ENROLL NOW

Next scheduled date: March 15th, 2026 @ 10:00 AM EDT

AI Security Assessments

BHIS can help identify and mitigate vulnerabilities unique to artificial intelligence systems, ensuring your organization deploys AI securely and responsibly.

bhis.co

BLACK HILLS
Information Security

https://www.youtube.com/@AISecurityOps

Section 1

LLM Security Introduction

What is an LLM?

- A very large deep learning AI model that has been trained on vast amounts of textual data

- Core functionality includes:
  - Natural Language Processing (NLP): The processing, understanding and generation of human language

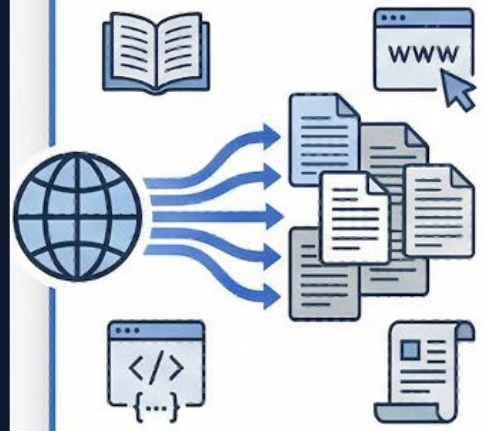
  - Text Generation: production of coherent and relevant text output

  - Language Understanding: the ability to learn and understand the relationship between words, grammar, and semantics, and respond appropriate.

FOUNDATION MODEL TRAINING

How LLMs Learn Language

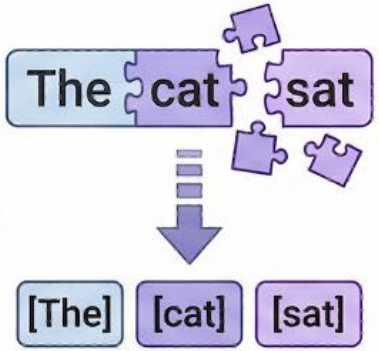
INTERNET TEXT



Trillions of Words

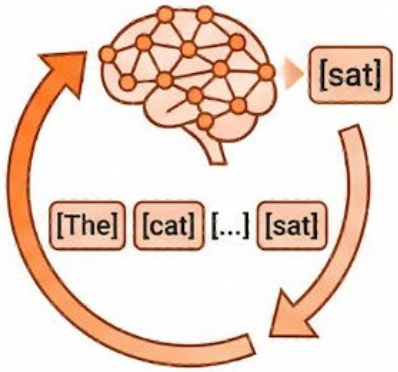
Books, Websites, Code, Papers

TOKENIZATION



Break Into Tokens

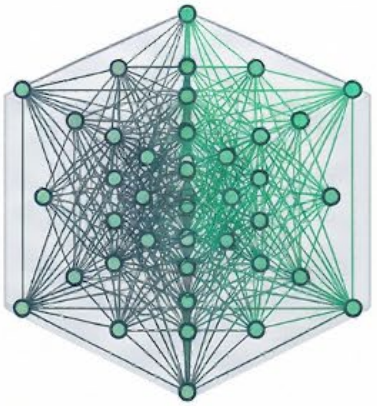
NEXT-TOKEN PREDICTION



Predict Next Token

Billions of iterations

MODEL WEIGHTS



Trained Model

1.8 Trillion Parameters

$63-100M+ | 90-100 Days | 25,000 GPUs | 50 GWh | Big Tech Only

Chatbot

- User Prompt: Input from user, typically via a chat-style interface
- System Prompt: Who the bot is, how it behaves, and what it will not do
  - Prepended onto User Prompt
- Optional Tools: APIs, scripts, or integrations the bot can use
- Optional Knowledge: Documents, runbooks, policies, reference materials, etc.
- Custom chatbot = system prompt + optional tools + optional knowledge

AI Agents

- An AI agent is a system that can perceive its environment, make decisions, and take actions autonomously to achieve a goal.

- Unlike traditional scripts or tools, an AI agent:

  - Can operate continuously (or semi-continuously)

  - Adapts based on feedback

  - Can plan, reason, and act without constant human input

LLM Safety versus Security

- Safety
  - Alignment
  - Bias and fairness
  - Harmful content
  - Confabulations / inaccurate information

- Security
  - Sensitive Information Disclosure
  - Excessive Agency
  - Training Data/Model Poisoning
  - Unbounded consumption

Section 2

OWASP Top 10 Deep Dives

OWASP Top 10 for LLM Applications

LLM01	Prompt Injection
LLM02	Sensitive Information Disclosure
LLM03	Supply Chain
LLM04	Data and Model Poisoning
LLM05	Improper Output Handling
LLM06	Excessive Agency
LLM07	System Prompt Leakage
LLM08	Vector and Embedding Weakness
LLM09	Misinformation
LLM10	Unbounded Consumption

LLM01:2025

Prompt Injection

User prompts alter LLM behavior to override instructions, bypass restrictions, or execute unintended actions - through direct or indirect attacks.

⚠ Vectors

- Direct: malicious user input overrides system prompt
- Indirect: injected content via RAG, web, or uploaded docs
- Multi-turn: injecting context across conversation history
- Multi-Modal: Embedding malicious prompts in images or video

Assess

- Jailbreaks, DAN prompts & role-play bypasses
- 'Ignore previous instructions' patterns
- Payloads hidden in files, URLs, or retrieved content

Defend

- Input/output validation and filtering
- Privilege separation
- System Prompt Security
- Human-in-the-Loop for High-Risk Actions
- A perfect solution is impossible currently

LLM02:2025

Sensitive Information Disclosure

LLMs inadvertently reveal confidential data including PII, credentials, business logic, or training data in their responses.

⚠ Risks

- PII or credentials in training data surfaces via prompts
- Business logic and internal process leakage
- Confidential or user data leaked

Assess

- Probe for PII via targeted completion attacks
- Ask the model about internal systems or processes
- Feed the model data and test what it retains or leaks

Defend

- DLP and output filtering on all responses
- Minimize sensitive data included in prompts/context
- Privacy-aware fine-tuning and data sanitization
- Ensure proper isolation of user data

LLM03:2025

Supply Chain

Vulnerabilities introduced through third-party models, datasets, plugins, or infrastructure in the LLM pipeline.

⚠ Vectors

- Compromised or backdoored pretrained models
- Malicious third-party plugins or tool integrations
- Poisoned datasets from public sources

Assess

- Audit all third-party model and plugin provenance
- Scan ML dependencies for known CVEs
- Verify model checksums and behavioral consistency

Defend

- SBOM (Software Bill of Materials) for AI components
- Verify model hashes before deployment
- Vet all third-party integrations

LLM04:2025

Data and Model Poisoning

Malicious manipulation of pre-training, fine-tuning, or embedding data causes the model to behave incorrectly or maliciously.

⚠ Vectors

- Backdoor triggers embedded via training/fine-tuning data
- Injection through scraped/crowdsourced datasets
- Poisoned embeddings in RAG knowledge bases
- Model performance degradation through corrupted training data
- Direction manipulation of model weights

Assess

- Test for behavioral anomalies with trigger phrases
- Probe fine-tuned models for backdoor activation
- Audit embedding stores for injected malicious content
- Investigate RBACs surrounding model and data

Defend

- Validate and vet all training and embedding data
- Monitor model behavior during training/tuning/deployment
- Grounded reasoning techniques to reduce confabulations

LLM05:2025

Improper Output Handling

Insufficient validation, sanitization, and handling of LLM-generated output enables downstream attacks like XSS, SQLi, and RCE.

⚠ Vectors

- LLM output rendered as HTML → XSS
- Output passed to shell commands → RCE
- Output used in database queries → SQLi

Assess

- Craft prompts that produce XSS/SQLi payloads in output
- Test whether output is sanitized before rendering
- Probe agentic pipelines for code execution via output

Defend

- Treat all LLM output as untrusted user input
- Sanitize and encode output before downstream use
- Never execute LLM-generated content directly

LLM06:2025

Excessive Agency

LLM-based systems granted too much autonomy take high-impact real-world actions beyond what is intended or safe.

⚠ Vectors

- Agents autonomously delete files, send emails, modify DBs
- Chained tool calls escalate to high-impact actions
- Prompt injection triggers unintended agentic behavior

Assess

- Prompt agent to perform privileged actions not directly available to the user
- Chain injections to escalate from read to write
- Test what actions execute without user confirmation

Defend

- Human-in-the-loop for all sensitive/irreversible actions
- Least privilege on all agent tool and API permissions
- Don't install OpenClaw and give it access to your work data

LLM07:2025

System Prompt Leakage

The system prompt, which potentially contains instructions, personas, and sensitive configuration data, is extracted by attackers through targeted prompting.

⚠ Vectors

- Prompt Injection methods
- Indirect inference by probing constraints
- Leakage via verbose error messages or debug output

Assess

- Attempt all known system prompt extraction techniques
- Probe behavioral constraints that reveal prompt content
- Test whether error states expose prompt fragments

Defend

- Design system prompts assuming they will be leaked
- Never store secrets (keys, PII) in system prompts
- Implement guardrails and system prompt protections to help mitigate the success of prompt injection attacks

LLM08:2025

Vector and Embedding Weaknesses

Security vulnerabilities in vector databases and embedding pipelines allow data poisoning, sensitive data extraction, and cross-tenant leakage.

⚠ Vectors

- Injecting malicious content into vector/embedding stores
- Cross-tenant data leakage in shared embedding DBs
- Embedding inversion attacks to recover significant amounts of source information

Assess

- Inspect the controls surrounding access to the vector stores
- Test for cross-user data leakage
- Probe embedding stores for sensitive data retrieval

Defend

- Strict access controls and tenant isolation on vector DBs
- Validate and sanitize all content before embedding
- Monitor retrieval results for suspicious behavior

LLM09:2025

Misinformation

LLMs generate false, misleading, or fabricated content - including confabulations, fake citations, and biased outputs - that users act upon.

⚠ Vectors

- Confident confabulations in legal, medical, financial domains
- Fabricated citations presented as authoritative
- Biased outputs reflecting training data imbalances

🔴 Assess

- Test for confident wrong answers in edge case domains
- Ask for citations and verify their existence
- Probe for bias in outputs about sensitive populations

🛡 Defend

- RAG grounding with verified, cited sources
- Human review gates for all high-stakes outputs
- Clearly communicate model limitations to end users

LLM10:2025

Unbounded Consumption

LLMs consume excessive resources (compute, tokens, API calls, costs) due to malicious or uncontrolled inputs, causing DoS and/or financial damage.

⚠ Vectors

- Recursive or deeply nested prompts exhaust compute
- Sponge attacks engineered to maximize token usage
- Automated scripts flooding APIs to inflate costs

Assess

- Send extremely long, recursive, or nested prompts
- Measure cost and latency under adversarial load
- Test rate limit enforcement and bypass techniques

Defend

- Rate limiting and per-user token caps
- Cost monitoring and anomaly alerting
- Timeout policies, circuit breakers, input length limits

Section 3

Red Team Methodology

How to approach LLM security testing systematically

The LLM Red Team Lifecycle

01 **Reconnaissance**

Map the surface: system prompt, plugins, RAG sources, API endpoints, user roles

02 **Threat Modeling**

What data does it touch? What actions can it take? What's the blast radius?

03 **Attack Planning**

Select relevant OWASP categories. Build a test matrix of payloads.

04 **Execution**

Iterate with varied prompts and roles. Document every result.

05 **Reporting**

Map findings to OWASP IDs, severity, and remediation guidance.

Threat Modeling an LLM App

Key Question	Why It Matters
What data does the LLM access?	Determines sensitivity of disclosure risk
Can it take real-world actions?	Assess blast radius for Excessive Agency
What plugins/tools are connected?	Plugin attack surface
Where does input come from?	Indirect injection vectors
Is output rendered or executed?	Insecure output handling
Who are the users?	Privilege escalation & authZ bypass risk

Section 4

Defensive Frameworks

Building secure LLM applications from the ground up

Defense-in-Depth for LLM Apps

Input Layer

- Prompt validation & sanitization
- Input length limits
- User role & intent classification

Model Layer

- System prompt hardening
- Fine-tuned refusal behaviors
- Output confidence thresholds

Output Layer

- Response filtering & DLP
- Output encoding for downstream use
- PII scrubbing before display

Integration Layer

- Least privilege for plugins & tools
- Human-in-the-loop for actions
- Audit logging of all LLM calls

Secure AI Development Lifecycle

📐 Design

- Threat model the LLM integration
- Define trust boundaries
- Scope model capability to need

🔨 Build

- Implement input/output guardrails
- Apply least privilege to tools
- Keep secrets out of system prompts

🧪 Test

- Run OWASP LLM Top 10 tests
- Red team before launch
- Automate adversarial regression tests

📡 Monitor

- Log all inputs & outputs
- Alert on anomalous patterns
- Continuously red team in production

Red Team Tools & Frameworks

Garak

LLM vulnerability scanner - jailbreaks, injections, data leaks

PyRIT

Microsoft's Python red-team tool for AI systems

LLMFuzzer

Fuzzing framework for prompt injection discovery

Burp Suite

Intercept & manipulate LLM API calls in web apps

PromptBench

Adversarial robustness evaluation for LLMs

Deepteam

LLM vulnerability scanner - jailbreaks, injections, data leaks

Key Takeaways

- ✓ LLMs introduce new vuln classes...but traditional application security still applies
- ✓ Prompt injection is the SQL injection of the AI era
- ✓ The integration layer is one of the biggest risk surface
- ✓ Least privilege, human oversight, and untrusted output by default
- ✓ Defense-in-depth across all layers of your LLM stack
- ✓ Red teaming is iterative - build it into your SDLC

Questions?