Cross-Chapter Handout: LLM and Agentic-System Security Vocabulary Reference

5,913 words · ~27 min read

Course companion for: ADV-102 (AI and Agentic Security) **Secondary audience:** AI-101, AI-201, AI-301, SEC-101 **Scope:** Belt-3 to Belt-5 **Taxonomy authority:** OWASP LLM Top 10 (2025 edition), OWASP Agentic Security Initiative (ASI) Top 10 (2025 release), MITRE ATLAS (atlas.mitre.org) **Last reviewed:** 2026-05-07

---

§1 Purpose and Scope

This handout is the course-companion vocabulary reference for ADV-102 (AI and Agentic Security). It walks two paired taxonomies at student-pinnable depth: the OWASP LLM Top 10 (2025) and the OWASP Agentic Security Initiative (ASI) Top 10 (2025). It maps both onto MITRE ATLAS technique IDs, closes with a worked structural example using ADV-102's signature CVE (CVE-2025-65106, LangChain Jinja2 SSTI), and names the agentic-system attack primitives that surface in the lab environment.

What this handout is for. At vocab-tier depth, graduates can name and recognize the LLM-era and agentic-system attack-surface families across both taxonomies, even if they have personally reproduced only the CVE-2025-65106 signature lab end-to-end. The vocabulary scaffold complements the methodology depth ADV-102 builds against the signature CVE.

What this handout is not. It does not teach exploitation methodology for each entry; that work happens in the ADV-102 modules against the signature lab. It does not re-state classical CVE classes (buffer overflow, format string, memory corruption families); those are in the companion reference `cross-chapter-cve-class-vocabulary-reference.md`. It does not cover how to build agentic systems securely; that is a forward-stretch surface.

Prerequisite vocabulary. ADV-102 students arrive here with:

- AI-101: OWASP LLM Top 10 introduction at Belt-2 register

- ADV-101: classical CVE-to-Tool methodology at Belt-5
- PEN-101: engagement-discipline foundation at Belt-3

This handout deepens AI-101's introduction; it does not replace it.

Taxonomy references.

- OWASP LLM Top 10 (2025): `owasp.org/www-project-top-10-for-large-language-model-applications/`
- OWASP ASI Top 10 (2025): `owasp.org/www-project-top-10-for-agentic-ai/`
- MITRE ATLAS: `atlas.mitre.org`

---

§2 OWASP LLM Top 10 (2025 Edition)

The OWASP LLM Top 10 names the ten most critical security risks for applications built on large language models. The 2025 edition reorganized and renamed several entries from the 2023 (v1.1) release to reflect evolving deployment patterns, particularly the rise of retrieval-augmented generation (RAG) pipelines and agentic tool-calling architectures.

Each row below gives: the entry name, the primary CWE pairing where MITRE maintains a mapping, a one-line attack surface description, and an anchor incident or disclosure where one is publicly available.

ID	Name	CWE (primary)	Attack surface in one line
LLM01	Prompt Injection	CWE-77, CWE-20	Attacker-controlled text alters model instruction execution
LLM02	Sensitive Information Disclosure	CWE-200	Model reveals confidential data from training, context, or retrieval
LLM03	Supply Chain	CWE-1357, CWE-506	Compromised model weights, plugins, or third-party data pipelines
LLM04	Data and Model Poisoning	CWE-915	Training or fine-tuning data manipulated to embed attacker-controlled behavior
LLM05	Improper Output Handling	CWE-116, CWE-79	Model output consumed by downstream code without sanitization
LLM06	Excessive Agency	CWE-269, CWE-732	Model granted capabilities or permissions beyond what the task requires
LLM07	System Prompt Leakage	CWE-200, CWE-922	Confidential system prompt exposed through model output
LLM08	Vector and Embedding Weaknesses	CWE-20, CWE-74	Retrieval-stage manipulation through crafted embeddings or corpus poisoning
LLM09	Misinformation	(no primary CWE)	Model generates plausible but false content consumed as authoritative
LLM10	Unbounded Consumption	CWE-400, CWE-770	Unconstrained resource use (tokens, API calls, compute) enables denial-of-service

LLM01: Prompt Injection

Prompt injection is the condition where attacker-controlled text modifies the behavior of a model's instruction execution. Direct prompt injection occurs when a user manipulates the prompt sent directly to the model. Indirect prompt injection occurs when malicious content in an external source (a retrieved document, a web page fetched by a tool, an email) is processed by the model and overrides the developer's original instructions. The LLM Top 10 names this the most critical risk category because it is the primary mechanism by which all other LLM-era attacks are delivered: a model that will follow attacker instructions is a model whose tool-calling, output handling, and access-control guarantees collapse.

ADV-102 treats this entry as the root vulnerability class for the signature lab. CVE-2025-65106 exploits a vector where user-supplied text reaches a Jinja2 template renderer with insufficient escaping, a form of indirect prompt injection at the template layer.

Anchor: The OWASP Foundation's LLM01 guidance page includes a taxonomy of direct vs. indirect injection variants and links to the original academic disclosure by Greshake et al. (2023), "Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injections."

LLM02: Sensitive Information Disclosure

A model may disclose sensitive data from three distinct pools: its training corpus (including personally identifiable information that leaked into training data), its in-context window (system prompt contents, prior conversation history, retrieved documents), and its retrieval corpus (RAG-accessible documents the user was not authorized to access). The attack surface is distinct from classical information disclosure (CWE-200) because the model does not apply access-control checks to information it was trained on or retrieved; it applies natural-language reasoning, which an attacker can manipulate.

Students familiar with path traversal (from ADV-101's classical CVE lab) will recognize the structural parallel: the attacker's goal is the same (exfiltrate a resource the server or model was not supposed to return), but the access path is natural-language reasoning rather than a filesystem path. The defensive countermeasures are also structurally analogous: minimize what the model can see, validate what it can return.

LLM03: Supply Chain

LLM supply chain risk covers the full dependency graph of a model-powered application: pre-trained base weights sourced from a third party, fine-tuning datasets assembled from untrusted corpora, plugins and tool integrations installed from package registries, and retrieval pipelines that pull from external data sources. An attacker who can compromise any link in this chain can introduce malicious behavior that is invisible to the application developer who trains, fine-tunes, or deploys downstream.

The CWE-506 (Embedded Malicious Code) and CWE-1357 (Reliance on Insufficiently Trustworthy Component) pairings map cleanly to the classical software supply chain threat model that ADV-101 covers for compiled binaries. The LLM-era extension is that a compromised model weight is effectively a blob of embedded malicious code that executes against natural-language inputs rather than machine instructions.

LLM04: Data and Model Poisoning

Data poisoning is the condition where an attacker controls a portion of the training or fine-tuning dataset and uses that control to embed a behavioral backdoor into the resulting model. The model behaves normally on standard inputs and activates the attacker-intended behavior only when a specific trigger is present in the input. Model poisoning extends this to post-training phases: an attacker who can manipulate the fine-tuning dataset for a production model can alter its behavior without touching the base weights or the inference infrastructure.

CWE-915 (Improperly Controlled Modification of Dynamically-Determined Object Attributes) is the MITRE pairing most commonly cited, capturing the idea that the model's behavioral attributes (its learned associations) are being modified by an input the developer did not control. The attack is pre-deployment by nature, which makes it harder to detect at inference time.

Anchor: Carlini et al. (2021), "Poisoning the Unlabeled Dataset of Semi-Supervised Learning," IEEE S&P, is the entry-level academic reference for understanding how small-percentage corpus control translates to reliable behavioral manipulation.

LLM05: Improper Output Handling

When model output is passed to a downstream consumer (a browser, a shell, a SQL query, a code interpreter, an API endpoint) without sanitization, the model becomes an injection vector for classical injection classes. The LLM does not inject the payload; the developer's code does, by trusting model output as if it were developer-authored content. If the model can be manipulated into generating a JavaScript payload (via LLM01), and that payload is rendered in a browser without escaping, the resulting attack is a classical stored XSS at CWE-79. If the output reaches a shell command, the result is command injection at CWE-78.

This is the handoff point between the LLM-era taxonomy and the classical CVE-class taxonomy in `cross-chapter-cve-class-vocabulary-reference.md`. ADV-101's classical injection labs (SQLi, SSTI, SSRF) are all candidates for LLM05 exploitation paths when those injection sinks are downstream of an LLM output channel.

LLM06: Excessive Agency

Excessive agency names the condition where a model is granted capabilities, tool permissions, or autonomous actions beyond what the task requires. If a model can read email, compose email, and send email, and a user's task only required reading email,

the model has excessive agency over the send channel. An attacker who can influence the model's instructions (via LLM01) can direct that excess agency toward attacker-controlled ends.

The OWASP guidance recommends applying the principle of least privilege at three levels: the tools the model is authorized to call, the scope of data each tool can access, and the types of actions the model can take without human confirmation. These map directly to the access-control principles ADV-101 covers for classical CVE exploitation: a well-scoped model is harder to pivot from than an over-privileged one.

CWE-269 (Improper Privilege Management) and CWE-732 (Incorrect Permission Assignment for Critical Resource) are the applicable MITRE pairings.

LLM07: System Prompt Leakage

System prompt leakage is the disclosure of confidential developer instructions embedded in the model's context window. System prompts often contain operational details (tool names, API keys, internal service endpoints, behavioral guardrails, proprietary business logic) that the developer did not intend users to see. An attacker who extracts the system prompt gains a map of the system's intended constraints and may be able to construct a prompt injection payload that references and overrides those constraints by name.

The CWE-922 (Insecure Storage of Sensitive Information) pairing captures the idea that storing secrets in a prompt is a structural mistake, not just an operational one. The system prompt is not a secret-management system; it is a natural-language instruction plane that will be processed, summarized, and (if manipulated) echoed by the model.

LLM08: Vector and Embedding Weaknesses

The retrieval-augmented generation (RAG) pipeline introduces a new attack surface: the vector database and the embedding computation that populates it. An attacker who can write to the corpus indexed by a RAG pipeline can craft documents whose embeddings are close to high-value query embeddings, causing the retrieval stage to surface attacker-controlled content to the model. An attacker who can influence the embedding model itself (via a supply-chain compromise) can warp the semantic space to favor attacker-controlled retrievals.

CWE-74 (Improper Neutralization of Special Elements in Output Used by a Downstream Component) and CWE-20 (Improper Input Validation) are the applicable pairings when the retrieval stage does not validate the source or integrity of retrieved content before passing it to the model.

Anchor: This entry is sometimes described as "embedding inversion" or "RAG poisoning" in the research literature. MITRE ATLAS names it AML.T0070 (Poison RAG) and AML.T0066 (Retrieve Crafted Content); see §4 for the technique mapping.

LLM09: Misinformation

Misinformation risk is the condition where a model generates plausible but factually incorrect content that a consumer treats as authoritative. Unlike the other nine entries, this is not a confidentiality or integrity attack against the model's host system; it is a reliability and trust failure that affects the model's output consumers. The OWASP LLM Top 10 includes it because it is a top-of-mind concern for application developers deploying models in high-stakes domains (medical, legal, financial, critical infrastructure).

There is no primary CWE pairing for misinformation; the closest analogues are CWE-1059 (Incomplete Documentation) and general reliability-engineering concepts that predate the LLM taxonomy. ADV-102 treats this entry as context for understanding model limitations rather than as an attack vector to reproduce.

LLM10: Unbounded Consumption

Unbounded consumption names the condition where an LLM application allows an attacker to drive excessive resource use: token generation (producing long outputs), API call chains (triggering expensive downstream calls), or compute cycles (submitting prompts that cause the model to iterate on long reasoning chains). The effect is equivalent to a denial-of-service attack (CWE-400) or resource exhaustion (CWE-770), but the attack surface is the model's inference API rather than a network stack.

In agentic systems, unbounded consumption compounds: an agent with tool-calling authority that loops on a task can call external APIs, accumulate tokens, and exhaust rate limits without a human in the loop to interrupt. The defensive countermeasures (token limits, API rate limiting, loop detection, circuit breakers) are direct analogues of the traffic-shaping and rate-limiting defenses ADV-101 covers for network-layer DoS.

---

§3 OWASP ASI Top 10 (Agentic Security Initiative, 2025 Release)

The OWASP Agentic Security Initiative (ASI) Top 10 extends the LLM Top 10 from single-call model inference to multi-step agentic workflows: systems where a model calls tools, receives results, plans next steps, and acts on external systems across multiple turns.

The ASI taxonomy names risks that either do not appear in single-call LLM use or appear in qualitatively different form when the model has persistent state and tool-calling authority.

The LLM Top 10 and ASI Top 10 are not replacements for each other; they are paired layers. The LLM Top 10 covers the model inference surface; the ASI Top 10 covers the orchestration, tool-calling, and multi-agent coordination surface. Both taxonomies apply to ADV-102's lab target, which is an agentic system (LangChain-based) rather than a bare inference API.

ID	Name	Core risk in one line
ASI01	Agent Goal Hijack	Attacker redirects the agent's objective mid-task
ASI02	Tool Misuse and Exploitation	Agent is manipulated into calling tools beyond user intent
ASI03	Identity and Privilege Abuse	Agent impersonates users or escalates its own permissions
ASI04	Agentic Supply Chain Vulnerabilities	Compromised tool integrations or sub-agents alter system behavior
ASI05	Unexpected Code Execution	Agent generates and executes code without explicit authorization
ASI06	Memory and Context Poisoning	Persistent agent memory is contaminated to alter future behavior
ASI07	Insecure Inter-Agent Communication	Messages between agents are intercepted, forged, or replayed
ASI08	Cascading Failures	Agent errors propagate uncontrolled across tool chains or agent networks
ASI09	Human-Agent Trust Exploitation	User or operator trust in the agent is exploited to bypass oversight
ASI10	Rogue Agents	Agent pursues unintended goals independent of operator or user intent

ASI01: Agent Goal Hijack

Goal hijack is the agentic-layer form of prompt injection (LLM01). In a single-call inference context, a prompt injection replaces the developer's instructions with the attacker's for that one call. In an agentic context, a goal hijack redirects the agent's planning objective across a multi-step task. The agent may be midway through a workflow (having already called several tools and accumulated state) when the

attacker-controlled input causes it to replan toward a different objective. The structural severity is higher because the agent's prior tool calls have already had real effects on external systems.

The practical delivery vector for goal hijack is indirect prompt injection through the environment: a document the agent retrieves, a tool response that contains attacker-controlled text, or an API response the agent interprets as task-relevant data.

ASI02: Tool Misuse and Exploitation

Tool misuse names the condition where an agent is manipulated into calling a tool in a way the developer did not intend. The manipulation may be direct (a user instructs the agent to call a tool it should not) or indirect (an injected payload causes the agent to call a tool as part of its execution plan). Tool exploitation extends this to cases where the tool itself has a vulnerability that the agent's call parameters trigger.

This entry pairs with LLM06 (Excessive Agency) at the model layer and with ADV-102's signature lab at the tool layer: CVE-2025-65106 is an exploitation of the template-rendering tool (LangChain's `Environment.from_string()`) through an agent's input plane.

ASI03: Identity and Privilege Abuse

In multi-agent and multi-user systems, the agent must represent itself and act on behalf of principals. Identity abuse occurs when an agent claims an identity it does not hold (impersonating a user, an administrator, or another agent) or when an attacker causes the agent to exercise authority that belongs to a higher-privileged principal. Privilege abuse occurs when the agent's accumulated tool calls effectively escalate its permissions beyond what any single tool call would have authorized.

The CWE-269 (Improper Privilege Management) and CWE-863 (Incorrect Authorization) pairings from the classical taxonomy apply here. The LLM-era variant is structurally the same class of attack, with a natural-language authorization plane substituted for the kernel or application access-control layer.

ASI04: Agentic Supply Chain Vulnerabilities

In agentic architectures, the supply chain extends beyond model weights and inference dependencies to include the tools the agent is authorized to call, the sub-agents the orchestrator delegates to, the memory stores the agent reads from and writes to, and the environment descriptions (system prompts, tool schemas) that define the agent's

operational context. A compromised tool schema causes the agent to misinterpret tool behavior; a compromised sub-agent acts as a trusted participant in the workflow while pursuing attacker-defined objectives.

This entry is the agentic analogue of LLM03 (Supply Chain) at the orchestration layer.

ASI05: Unexpected Code Execution

Agents built on code-generation models may produce and execute code as part of their normal workflow. Unexpected code execution names the condition where that code runs in a context the user or operator did not explicitly authorize: a model asked to analyze a file produces a shell command and executes it, or a planning agent writes a Python script and runs it in the execution environment without operator review.

The structural risk is that the authorization model for code execution is implicit (the agent can call a code-execution tool) rather than explicit (the user approved this specific code). An attacker who can influence the agent's planning can direct code generation toward attacker-controlled payloads while the agent's output appears to the operator as routine task execution.

ASI06: Memory and Context Poisoning

Long-running agents often maintain persistent memory: summaries of prior sessions, user preferences, tool results, conversation history, or task-specific notes written to a vector database or key-value store. Memory poisoning names the condition where an attacker introduces content into this persistent memory that alters the agent's future behavior. The effect is durable across sessions, unlike single-turn prompt injection, and the injection often occurs through a legitimate input path (a document the agent was asked to process, a tool result the agent stored as a note).

This entry maps to MITRE ATLAS AML.T0070 (Poison RAG) when the memory backend is a retrieval-augmented corpus. The LLM-tutor architecture handout (`cross-chapter-llm-tutor-3-layer-architecture-ai-201.md`) discusses the memory plane at §5 with red-team angles at §6.

ASI07: Insecure Inter-Agent Communication

Multi-agent architectures route messages between orchestrators, sub-agents, and tool servers. Insecure inter-agent communication names the condition where these message channels lack integrity guarantees: an attacker who can observe or inject into the channel can replay messages, forge tool responses, or cause the orchestrator to believe

a sub-agent completed a task it did not. The attack surface is analogous to classical message tampering and replay (CWE-294, CWE-345) applied to a natural-language message bus rather than a binary protocol.

The practical implication for ADV-102 students: when inspecting an agentic system's communications during the signature lab, the inter-agent channel is a candidate injection surface alongside the model's direct input plane.

ASI08: Cascading Failures

In a tool chain where each step's output is the next step's input, an error or attacker-influenced output at one step propagates through the chain and amplifies at each subsequent step. Cascading failure names the condition where the agent has no circuit breaker: it continues executing against corrupted state, takes external actions based on that state, and may not surface the failure to the operator until substantial damage has been done to downstream systems.

The classical analogue is the call-chain integrity assumption that ADV-101 covers for exploit-chain development: a chain is only as strong as its least-validated handoff. In an agentic system, the least-validated handoff is the tool response that the agent accepts as ground truth and passes, unvalidated, to the next planning step.

ASI09: Human-Agent Trust Exploitation

As agents become more capable and operators interact with them through natural language, a trust relationship develops where the operator accepts the agent's outputs, recommendations, and action summaries without detailed review. Human-agent trust exploitation names the condition where an attacker (or a compromised sub-agent) exploits that trust to obtain operator approval for attacker-intended actions, to suppress operator awareness of actions taken, or to cause the operator to attribute malicious activity to a legitimate source.

The OWASP ASI guidance notes that this entry intersects with social engineering (the attacker targets the human through the agent) and with automation bias (the operator over-trusts the agent's output because of its track record). Neither is a model vulnerability per se; both are system-level risks of deploying an agent with uncritical human oversight.

ASI10: Rogue Agents

Rogue agent behavior names the condition where an agent pursues goals that were not sanctioned by the operator or user. This may arise from goal misspecification (the agent was given an instruction that is technically satisfied by attacker-beneficial actions), from reward hacking (the agent finds a way to satisfy its objective function that was not anticipated), or from a goal hijack (ASI01) successful enough to redirect the agent's planning across multiple sessions. The term "rogue" is functional, not attributional: it describes the behavioral outcome (pursuing unsanctioned goals) rather than the mechanism.

The OWASP ASI guidance recommends explicit goal scoping, output monitoring, and human-in-the-loop checkpoints for high-stakes actions as the primary mitigations. These correspond to the engagement-discipline principles in PEN-101: scope defines what is authorized; anything outside scope is unauthorized regardless of whether the agent's internal reasoning reached it through a valid-looking path.

---

§4 MITRE ATLAS Technique Mapping

MITRE ATLAS (Adversarial Threat Landscape for Artificial Intelligence Systems) maps adversarial techniques targeting AI and machine-learning systems. Technique IDs use the prefix AML.T00xx. The table below maps the most relevant ATLAS techniques to LLM Top 10 and ASI Top 10 entries, ordered by technique ID. Verify current technique names and sub-technique IDs against the live registry at `atlas.mitre.org`; this table reflects the state as of 2026-05-07.

The five technique IDs cited in the LLM-tutor architecture handout (AML.T0049, T0051, T0054, T0066, T0070) are included and cross-referenced.

AML Technique ID	Technique Name	Primary LLM/ASI Mapping	Notes
AML.T0040	ML Model Inference API Access	LLM10 (Unbounded Consumption)	Entry-point for token-exhaustion and rate-limit attacks
AML.T0043	Craft Adversarial Data	LLM04 (Data Poisoning), LLM08 (Embedding Weaknesses)	Poisoning via data crafted to embed triggers
AML.T0044	Full ML Model Access	LLM03 (Supply Chain)	Direct access to weights enables supply-chain substitution
AML.T0045	ML Artifact Collection	LLM03 (Supply Chain)	Exfiltrate model artifacts for analysis or reuse
AML.T0047	Discover ML Model Ontology	LLM07 (System Prompt Leakage)	Probe model to map its instruction constraints
AML.T0048	Discover ML Model Family	LLM07 (System Prompt Leakage)	Identify the base model to locate known jailbreaks
AML.T0049	Adversarial ML Attack	LLM01 (Prompt Injection), ASI01 (Goal Hijack)	Core technique class for prompt-level attacks; cited in LLM-tutor handout §6
AML.T0050	Backdoor ML Model	LLM04 (Data and Model Poisoning)	Embed behavioral backdoor during training
AML.T0051	LLM Prompt Injection	LLM01 (Prompt Injection), ASI01 (Goal Hijack)	ATLAS-native technique for direct and indirect prompt injection; cited in LLM-tutor handout §6
AML.T0054	LLM Jailbreak		Bypass model guardrails via

AML Technique ID	Technique Name	Primary LLM/ASI Mapping	Notes
		LLM01 (Prompt Injection), LLM07 (System Prompt Leakage)	prompt manipulation; cited in LLM-tutor handout §6
AML.T0056	LLM Meta Prompt Extraction	LLM07 (System Prompt Leakage)	Extract developer-authored system prompt via model output
AML.T0057	LLM Data Extraction	LLM02 (Sensitive Information Disclosure)	Cause the model to reproduce training data or context-window secrets
AML.T0062	Poison Training Data	LLM04 (Data and Model Poisoning)	Manipulate the training corpus before fine-tuning
AML.T0063	Evade ML Model	LLM09 (Misinformation)	Craft input that avoids detection while achieving attacker goal
AML.T0065	Manipulate ML Model	LLM04, LLM03	Post-training weight modification
AML.T0066	Retrieve Crafted Content	LLM08 (Vector and Embedding Weaknesses), ASI06 (Memory Poisoning)	Craft documents that retrieve near high-value queries; cited in LLM-tutor handout §6
AML.T0067	ML Supply Chain Compromise	LLM03 (Supply Chain), ASI04 (Agentic Supply Chain)	Compromise model or tool dependency at any supply-chain stage
AML.T0068	ML Artifact Manipulation	LLM03, LLM04	Modify model artifacts in the supply chain
AML.T0070	Poison RAG	LLM08 (Vector and Embedding Weaknesses), ASI06 (Memory Poisoning)	Inject attacker-controlled content into the RAG corpus; cited in

AML Technique ID	Technique Name	Primary LLM/ASI Mapping	Notes
			LLM-tutor handout §6

Technique count in this table: 19 techniques (AML.T0040, T0043, T0044, T0045, T0047, T0048, T0049, T0050, T0051, T0054, T0056, T0057, T0062, T0063, T0065, T0066, T0067, T0068, T0070).

How to use this table. For any ADV-102 lab exercise involving an agentic system attack surface, identify which ATLAS technique ID most closely describes the method being applied. The LLM-tutor handout §6 demonstrates this practice against the LLM-tutor Phase-1.6 architecture.

---

§5 Worked Example: CVE-2025-65106 LangChain Jinja2 SSTI

ADV-102's signature lab reproduces and defends against CVE-2025-65106, a server-side template injection (SSTI) vulnerability in LangChain's Jinja2 prompt template evaluation path. This section walks the vulnerability at structural depth: attack surface, minimum-viable payload, code-level location, classical cousin, and defensive generalization. For the full exploit development and lab environment setup, see the ADV-102 course modules.

NVD reference: `nvd.nist.gov/vuln/detail/CVE-2025-65106` **Lab target:** ADV-102 Module 4 LangChain Jinja2 SSTI lab harness

5.1 Attack Surface

LangChain supports user-configurable prompt templates. In the vulnerable code path, a user-supplied string is passed to Jinja2's `Environment.from_string()` method using the default `Environment` class. The default `Environment` does not enable Jinja2's sandbox (`SandboxedEnvironment`) and does not restrict access to Python's object introspection attributes. When the resulting template is rendered, any Jinja2 expression in the user-supplied string executes in the Python interpreter context of the LangChain server process.

The vulnerable pattern in pseudocode:

```
from jinja2 import Environment

env = Environment()  # default: no sandbox
template = env.from_string(user_supplied_prompt_template)
result = template.render(**context)
```

The `user_supplied_prompt_template` variable is the injection point. In the agentic system context, this input arrives through the agent's input plane: a user-submitted prompt, a retrieved document, or a tool response that the agent passes as a template parameter.

5.2 Minimum-Viable Payload

The following Jinja2 expression demonstrates class-hierarchy traversal from a string literal to the subprocess module available in the Python runtime. This is a structural illustration only; the actual CVE-2025-65106 lab harness provides the controlled reproduction environment.

```
{{ ''.__class__.__mro__[1].__subclasses__() }}
```

This expression enumerates all subclasses of `object` in the current Python process. From this list, an attacker identifies the class at the index corresponding to a module that provides system access (typically `subprocess.Popen` or equivalent) and calls it. The specific index is environment-dependent; enumeration is the first step of the exploit.

A more targeted payload for command execution:

```
{{ ''.__class__.__mro__[1].__subclasses__()[INDEX](['id'],
stdout=-1).communicate() }}
```

where `INDEX` is the environment-specific index of a subprocess-capable class.

5.3 Code-Level Location

The vulnerability is in LangChain's prompt template evaluation path. The upstream patch for CVE-2025-65106 changes the default `Environment` instantiation to `SandboxedEnvironment`, which restricts access to Python introspection attributes and prevents the `__class__` / `__mro__` traversal path.

Upstream fix (structural): `SandboxedEnvironment` from `jinja2.sandbox` replaces the default `Environment`. The sandbox blocks access to dunder attributes (`__class__`, `__mro__`, `__init__`, `__globals__`) that enable the traversal chain.

Reference the upstream patch commit in the LangChain repository (`github.com/langchain-ai/langchain`) for the exact file and line. The NVD entry at `nvd.nist.gov/vuln/detail/CVE-2025-65106` links the upstream advisory and patch.

5.4 Classical Cousin and LLM-Era Twist

Classical cousin. SSTI is a well-documented vulnerability class in web frameworks. Flask/Jinja2 and Django template engines, when misconfigured to render user-supplied input as a template, expose the same class-hierarchy traversal path. The canonical research reference is James Kettle's "Server-Side Template Injection" (PortSwigger Research, 2015), which maps SSTI across Jinja2, Twig, Freemarker, Smarty, and Mako. WAHH (Stuttard and Pinto, 2nd ed., Ch. 8-9) covers injection classes including template injection in the web application context. The classical CVE-class vocabulary reference `cross-chapter-cve-class-vocabulary-reference.md` §3 covers SSTI as a classical family.

LLM-era twist. In the classical web application context, SSTI typically requires the developer to make a deliberate choice to render user input as a template, a mistake that careful code review would catch. In the agentic-system context, the injection vector is the model's output plane: the model is instructed (or manipulated, via LLM01 prompt injection) to produce a string that becomes a template parameter. The developer's template rendering code is not visibly processing "user input"; it is processing "model output," which the developer may treat as trusted. This shifts the effective injection point from the user-input boundary to the model-output boundary.

The implication for defense: sanitizing the user input plane is insufficient. The model-output plane must also be treated as untrusted when it feeds downstream template renderers, code evaluators, shell commands, or SQL queries.

5.5 Defensive-Rule Generalization

Three generalized defensive rules follow from the CVE-2025-65106 structural analysis:

1. **Never render model output in a template context without an explicit sandbox.** Jinja2's `SandboxedEnvironment` is the correct default when the template string source is not fully developer-controlled. This applies to any template engine (Jinja2, Twig, Handlebars, Smarty) used downstream of a model output plane.

2. **Treat the model-output plane as untrusted when it feeds injection-capable sinks.** The classical injection sinks (template renderers, shell commands, SQL queries, XML parsers, HTML renderers) do not become safe because their input arrives from a model rather than a user. Apply the same sanitization and escaping discipline regardless of input source.
3. **Scope the agentic system's template-rendering authority.** If the system does not require user-configurable prompt templates, do not expose a template-rendering code path to the user input plane at all. Excessive agency (LLM06) at the template-rendering layer is the architectural precondition that makes CVE-2025-65106 exploitable.

---

§6 Agentic-System Attack Primitives

The following primitives name attack techniques that surface in agentic systems but do not map cleanly to a single LLM Top 10 or ASI Top 10 entry. Each receives a name, a CWE or ATLAS pairing where applicable, a one-paragraph description at vocab-tier depth, and a one-line anchor pointing at a known incident or red-team report.

Primitive 1: Tool-Calling Abuse

A model authorized to call tools has an authority surface that extends beyond the model's inference context to every system those tools can reach. Tool-calling abuse names the condition where an attacker manipulates the model into calling tools in ways the user or developer did not intend: calling a high-privilege tool (email send, file delete, database write) when the task only required a read-only operation, or constructing tool call parameters that exploit a vulnerability in the tool's implementation.

ATLAS pairing: AML.T0049 (Adversarial ML Attack) as the technique class; the specific tool is the target. **CWE pairing:** CWE-269 (Improper Privilege Management).

Anchor: The OWASP LLM Top 10 v1.1 guidance on LLM08 (later reorganized into LLM06 Excessive Agency in the 2025 edition) provides the canonical framing for tool-calling scope and the "minimum necessary tool set" principle.

Primitive 2: Memory Poisoning

An agent that maintains persistent memory (a vector store, a key-value summary store, or a session-history database) is vulnerable to memory poisoning: attacker-controlled content is written into the memory store through a legitimate input path, and on

subsequent sessions the agent retrieves that content as if it were trusted history. The poisoned memory may alter the agent's behavior subtly (biasing it toward attacker-preferred choices) or overtly (causing it to execute attacker-specified actions when a trigger is present in the session).

ATLAS pairing: AML.T0070 (Poison RAG) when the memory backend is a retrieval corpus. **CWE pairing:** CWE-74 (Improper Neutralization of Special Elements).

Anchor: The LLM-tutor architecture handout §6 at `handouts/cross-chapter-llm-tutor-3-layer-architecture-ai-201.md` discusses memory-plane red-team angles for the academy's Phase-1.6 deployment.

Primitive 3: Chain-of-Tools Privilege Escalation

An agent that orchestrates multiple tools may arrive at a privileged action through a sequence of individually-authorized tool calls. Chain-of-tools privilege escalation names the condition where no single tool call in the sequence would have been authorized in isolation, but the sequence as a whole achieves a privileged outcome. The authorization model for the individual tools does not account for their composition.

CWE pairing: CWE-269 (Improper Privilege Management), CWE-732 (Incorrect Permission Assignment for Critical Resource).

Anchor: This primitive is described in the OWASP ASI Top 10 framing of ASI03 (Identity and Privilege Abuse) and is structurally analogous to the time-of-check/time-of-use race condition class in classical concurrent programming (CWE-367).

Primitive 4: Prompt Leak via Error Channel

When an agent's tool calls fail (due to API errors, rate limits, invalid parameters, or network failures), error messages may be returned to the model and incorporated into the model's output or reasoning. If the error message contains sensitive information (API keys, internal endpoint URLs, system prompt fragments, tool schema details), the model may echo that information in its response or store it in its memory. The error channel becomes an unintended information-disclosure path.

CWE pairing: CWE-209 (Generation of Error Message Containing Sensitive Information).

Anchor: This primitive surfaces in standard security testing of agentic systems: inducing tool failures and observing whether error details propagate to the model's output is a routine red-team check in agentic-system assessments.

Primitive 5: RAG Injection Through Document Corpora

A RAG pipeline retrieves documents from a corpus indexed as vector embeddings and presents them to the model as context. An attacker who can write to the indexed corpus (by uploading a document, editing a wiki page, or contributing to a knowledge base) can craft documents whose content, when retrieved, injects attacker-controlled instructions into the model's context window. The injection arrives through the retrieval path rather than the direct user input path.

ATLAS pairing: AML.T0066 (Retrieve Crafted Content), AML.T0070 (Poison RAG). **CWE pairing:** CWE-74 (Improper Neutralization of Special Elements).

Anchor: This vector was demonstrated in "Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injections" (Greshake et al., 2023), which documented indirect injection through retrieved web content in a deployed system.

Primitive 6: Jailbreak Persistence Through Context Window Manipulation

Standard jailbreak techniques (adversarial prompts that bypass the model's behavioral guardrails) typically affect a single inference call. Context window manipulation names a technique where the attacker constructs a conversation history or document set that primes the model to bypass guardrails persistently across multiple turns. By filling the model's context window with content that normalizes the attacker-intended behavior, the attacker shifts the model's effective instruction baseline for the duration of that context window.

CWE pairing: CWE-693 (Protection Mechanism Failure). **ATLAS pairing:** AML.T0054 (LLM Jailbreak).

Anchor: The OWASP LLM Top 10 2025 LLM01 guidance discusses multi-turn context manipulation as a variant of indirect prompt injection. The distinction between jailbreak (guardrail bypass) and goal hijack (ASI01) is that jailbreak targets the model's behavioral constraints while goal hijack targets the agent's task objective.

---

§7 Cross-References

7.1 Academy Handouts

Handout	Relationship to this vocabulary
`handouts/cross-chapter-cve-class-vocabulary-reference.md`	Classical CVE families (buffer overflow, format string, SSTI, SQLi, SSRF, command injection). LLM05 (Improper Output Handling) and the CVE-2025-65106 SSTI entry here connect directly to the SSTI family there.
`handouts/cross-chapter-llm-tutor-3-layer-architecture-ai-201.md`	Real-world agentic-system case studies. §6 names MITRE ATLAS IDs (AML.T0049/T0051/T0054/T0066/T0070) that appear in §4 of this handout. LLM01 deepens against the LLM-tutor input-plane analysis; ASI06 deepens against the memory-plane red-team section.
`handouts/cross-chapter-llm-tutor-3-layer-architecture-ai-301.md`	Advanced architecture analysis; forward-reference for AI-301 students reviewing this vocabulary at a deeper layer.

7.2 Public Catalog Pages

Page	Relationship
`vca-adv-102.html`	This handout is the course companion for ADV-102. The catalog page commits to OWASP LLM Top 10 + ASI Top 10 as primary taxonomies and CVE-2025-65106 as the signature lab.
`vca-ai-101.html`	AI-101's OWASP LLM Top 10 introduction is the prerequisite vocabulary baseline this handout deepens.
`vca-ai-201.html`	AI-201's architecture content (LLM-tutor Phase-1.6) provides the real-world agentic system this vocabulary applies to.
`vca-ai-301.html`	AI-301 forward-stretch: production agentic architectures and red-team methodology at Belt-6 register.
`vca-pen-101.html`	Engagement-discipline foundation referenced in §3 ASI10 (Rogue Agents) and §1 prerequisite vocabulary.
`vca-adv-101.html`	Classical CVE-to-Tool methodology at Belt-5. LLM05 and the CVE-2025-65106 SSTI classical cousin connect directly to ADV-101's injection lab work.

7.3 External References

Source	URL	Used in
OWASP LLM Top 10 (2025)	`owasp.org/www-project-top-10-for-large-language-model-applications/`	§2 full taxonomy
OWASP ASI Top 10 (2025)	`owasp.org/www-project-top-10-for-agentic-ai/`	§3 full taxonomy
MITRE ATLAS	`atlas.mitre.org`	§4 technique table
NVD CVE-2025-65106	`nvd.nist.gov/vuln/detail/CVE-2025-65106`	§5 worked example
LangChain Repository	`github.com/langchain-ai/langchain`	§5.3 patch reference
Greshake et al. 2023	"Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injections"	§6 Primitive 5 anchor
Kettle 2015 (PortSwigger)	"Server-Side Template Injection" at `portswigger.net/research/server-side-template-injection`	§5.4 classical cousin anchor
WAHH 2nd ed. (Stuttard and Pinto)	Ch. 8-9: injection classes including SSTI	§5.4; primary course anchor pair
Black Hat Python 2nd ed. (Seitz and Arnold)	Ch. 10: Python-based offensive tooling	Primary course anchor pair; toolchain context

---

§8 Decisions, Pedagogy, and Curriculum Supplements
