AI Security Workshop

Threat Modeling for LLM apps

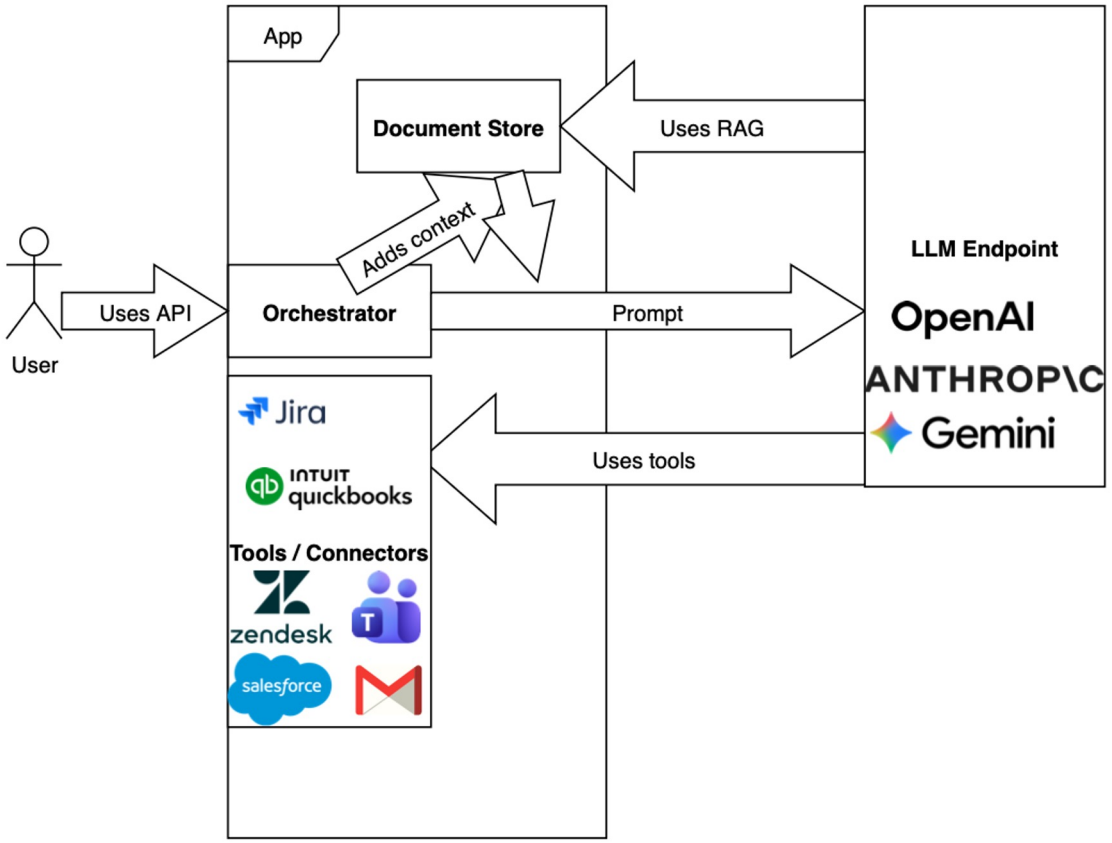
From Risk Categories to Attacker Behavior

- **Why threat modeling matters**
  - Real attacks chain **inputs, context, tools, and permissions**
  - The same OWASP issue can be reached in very different ways
  - Defenses fail at **trust boundaries**, not category labels

- We went over many types of risks – let's see why and when attackers use them

LLM-based applications

- **User / Client**: Chat UI, API, or embedded assistant
- **Orchestrator (control plane)**
  - Prompt construction & instruction hierarchy
  - Policy enforcement (authZ, data access, safety)
  - Model routing and tool-selection logic
- **LLM Endpoint**
  - Hosted or self-managed model inference
  - Executes with system / developer instructions
- **Retrieval (RAG)**
  - Document sources, embeddings, vector index, ranking
  - Supplies scoped runtime context
- **Tools / Connectors**
  - Databases, CRM, ticketing, email, files, code execution
  - Produce real-world side effects
- **Telemetry Plane**
  - End-to-end “AI turn” tracing
  - Links prompt → retrieval → tool calls → model output
  - Enables detection, audit, and incident response



OWASP Top 10 ↔ MITRE ATLAS ™ examples

- **OWASP** describes *what can go wrong* (risk categories)
- **ATLAS** describes *how attackers do it* (tactics & techniques)

- **Prompt Injection (LLM01)**
  - **ATLAS**: *AML.T0051 - LLM Prompt Injection*
  - **Sub-techniques:**
    - AML.T0051.000 – *Direct* (attacker prompt)
    - AML.T0051.001 – *Indirect* (via external data like RAG)
    - Also linked to AML.T0054 – LLM Jailbreak

- **Sensitive Information Exposure (LLM02)**
  - **ATLAS:** *AML.T0057 – LLM Data Leakage* (models leaking training or sensitive info)
  - Also related to exfiltration and model inversion techniques

Why LLM apps change threat modeling

- **LLMs blur traditional trust boundaries**
  - **All inputs arrive as text**: user input, system instructions, retrieved data
  - No native separation between trusted and untrusted content

- **Control flow becomes data-driven**
  - Application behavior is influenced by natural language, not fixed logic
  - Retrieved or user-supplied text can affect decisions at runtime

- **Text can trigger real-world actions and bypass security controls**
  - Tool use turns model output into API calls, database queries, and file operations
  - Failures can have direct system impact, not just bad answers

MITRE ATLAS ™

- **What it is**
  - **Adversarial Threat Landscape for Artificial Intelligence Systems**
  - Focuses on *real-world* attacks against AI (training, inference, deployment)

- **What it is not**
  - Not a step-by-step threat modeling methodology
  - Does not replace STRIDE / PASTA / OCTAVE

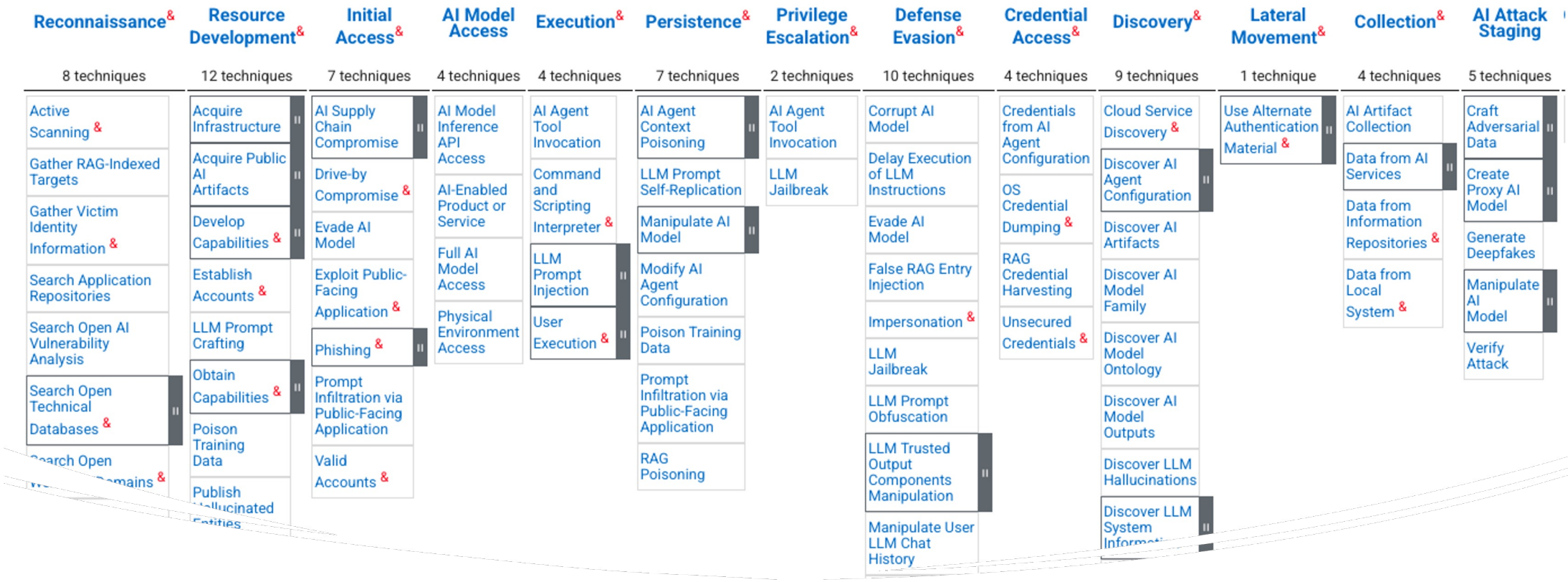
- Enumerates **AI-specific adversary goals, tactics** and **techniques**
- Helps identify attack surfaces unique to AI
- Enables systematic coverage of AI threats during modeling
- **Enables cross-team coordination (red team ↔ SOC ↔ engineering) with shared terms**

Using MITRE ATLAS ™ in threat modeling

- **Where ATLAS fits**
  - Input to your threat modeling process, not the process itself
  - Used after system architecture & asset identification

- **How teams apply it**
  - Map AI components to ATLAS ™ tactics & techniques
  - Identify realistic attacker goals per lifecycle stage
  - Prioritize threats based on impact and exposure

- **Typical outputs**
  - AI-specific threat list
  - Mapped mitigations and security controls
  - Gaps in existing controls



Using MITRE ATLAS™ in threat modeling

Like other MITRE matrices, ATLAS defines a matrix, progressing from left to right, demonstrating how an AI attack looks like during its different stages

TTPs in MITRE ATLAS™

- **Tactic** = attacker *goal*
  - e.g., **Collect** sensitive info, **Exfiltrate**, **Impact**
- **Technique** = attacker *approach* to reach the goal
  - e.g., **Prompt Injection**, **Data Poisoning**, **Model Extraction**
- **Procedure** = how that technique is executed **in your system**
  - your app's exact steps, paths, and payloads

- Example (LLM app)
  - Tactic: Execution
  - Technique: LLM Prompt Injection
  - Procedure: attacker crafts an input that causes the assistant to override instructions and pull extra data via retrieval/tool calls

Trust Boundaries & Control Points

- Most AI failures occur when **data or control crosses a boundary without enforcement**

- **Primary control points**
  - Input validation & instruction hierarchy enforcement
  - Retrieval authorization & query constraints
  - Tool allowlists, scopes, and approval gates
  - Output filtering & post-processing
- **Key trust boundaries**
  - **User → Orchestrator**
    - User input cannot be trusted and must be validated and constrained.
  - **Retrieval → Orchestrator**
    - Retrieved documents may contain sensitive or malicious content and must be access-controlled.
  - **Tools → Orchestrator**
    - Tool results can change systems or data and must be explicitly authorized and checked.
  - **LLM → Orchestrator**
    - Model output is unpredictable and must never enforce security decisions.

ATLAS Tactics: (1) Reconnaissance

- Adversaries gather information about AI systems to plan and tailor future attacks.

- Enables attackers to:
  - Identify AI capabilities, models, and deployment patterns
  - Target specific AI artifacts or services

- How do we protect against this?
  - Reduce exposed signal
  - Harden public & semi-public surfaces

ATLAS Tactics: (2) Resource Development

- Adversaries acquire, create, or compromise resources needed to execute AI-enabled attacks.
- AI attacks aren't as different here – resources are similar to other kind of cloud-based attacks
  - **AI artifacts:** models, datasets, poisoned weights
  - **Infrastructure:** cloud workspaces, domains, serverless, hardware
  - **Accounts & access:** developer platforms, model hubs, SaaS AI tools
  - **Capabilities:** adversarial attack code, prompt tooling, deepfake pipelines
- Heavy use of public platforms (Hugging Face, Colab, Copilot, ChatGPT)

ATLAS Tactics: (3) Initial Access

- Adversaries obtain their first foothold in an AI-enabled system: cloud, mobile, or edge by abusing AI-specific and traditional entry vectors.

- Access can mean many things, mostly similar to traditional applications
  - Influencing a model
  - Evading an AI model
  - Using a valid account or Phishing
  - Exploiting a public-facing application
- AI attacks aren't as different here – resources are similar to other kind of cloud-based attacks
  - **AI artifacts:** models, datasets, poisoned weights
  - **Infrastructure:** cloud workspaces, domains, serverless, hardware
  - **Accounts & access:** developer platforms, model hubs, SaaS AI tools
  - **Capabilities:** adversarial attack code, prompt tooling, deepfake pipelines
- Heavy use of public platforms (Hugging Face, Colab, Copilot, ChatGPT)

ATLAS Tactics: (4) AI Model Access

- Adversaries obtain some level of access to an AI model, direct or indirect - to **probe it, influence it, or use it as an attack surface**.
- **Access levels (core idea)**
  - **Inference-only access:** public or authenticated APIs
  - **Product-mediated access:** copilots, agents, AI-enabled apps
  - **Physical environment access:** sensors, cameras, microphones
  - **Full model access:** weights, architecture, training pipeline
  - Even minimal access (API-only) is sufficient for many attacks
- Most incidents involve **API or product-mediated access**, not full model theft

ATLAS Tactics: (5) Execution

- Execution in AI systems is often **indirect execution**, where attacker-controlled *intent* or *logic* is executed via a trusted intermediary (model, agent, tool, or human).
- Adversaries cause **attacker-controlled actions or logic** to be executed, often via prompt injection into trusted AI tools, agents, or users.

- **How execution happens in AI systems**
  - **LLM prompt injection** → model coerced into issuing commands or generating executable payloads
  - **AI agent tool invocation** → model triggers real actions (API calls, scripts, workflows)
  - **Interpreter / sandbox abuse** → code executed via notebooks, plugins, or "safe" runtimes
  - **User-mediated execution** → AI output convinces a human to run code

- AI systems blur the line between *code generation* and *code execution*

ATLAS Tactics: (5) Execution - Real-world Examples

- **MathGPT: Prompt → Code Execution**
  - Prompt injection coerced the system into emitting code that was **automatically passed to an interpreter**.
  - No sandbox escape was required—the model was already wired to execute outputs.
  - Key failure: **model output treated as trusted code**.

- **Google Colab: Shared Runtime Abuse**
  - Attackers leveraged Colab notebooks to run **arbitrary Python** at scale.
  - Enabled crypto-mining, malware staging, and lateral movement via cloud credentials.
  - Key failure: **execution environments optimized for convenience, not adversarial use**.

- **AI Coding Assistants: Rules / Config Backdoors**
  - Malicious instructions embedded in rules files, prompts, or repo context.
  - Assistant generated **plausible, trusted code** with hidden backdoors.
  - Execution happened when developers ran or merged the code.
  - Key failure: **human trust as the execution boundary**.

ATLAS Tactics: (6) Persistence

- Adversaries maintain long-term influence or access by embedding themselves into **AI artifacts, agent state, or data**, surviving restarts, updates, and credential changes.

- **Agent context poisoning** → malicious instructions persist across sessions
- **Model manipulation** → backdoored or fine-tuned models retain attacker behavior
- **Training data poisoning** → future retrains reintroduce malicious behavior
- **RAG poisoning** → hostile content embedded in indexed knowledge sources
- **Public-facing prompt infiltration** → attacker content stored and later reused

ATLAS Tactics: (7) Privilege Escalation

- Adversaries expand their authority by coercing AI systems to act with **permissions higher than originally granted**.

- **AI Agent Tool Invocation**
  - Tools execute without **user-context authorization checks**
  - Model acts with **system or service-account privileges**
  - Result: model authority > user authority
- **LLM Jailbreak**
  - Policy and role boundaries bypassed at the model layer
  - Becomes privilege escalation when combined with tools lacking authz
  - Quite theoretical at this point, no known incidents so far

ATLAS Tactics: (8) Defense Evasion

- Adversaries deliberately shape inputs, behavior, or context to **avoid detection by AI-enabled security systems** and monitoring.

- How attackers evade AI defenses
  - **Evade AI models** → adversarial examples that slip past classifiers
  - **LLM jailbreaks** → bypass safety filters and policy checks
  - **Prompt obfuscation** → encode or fragment malicious intent
  - **Trusted output manipulation** → abuse citations, summaries, or “verified” sections
  - **False RAG injection** → seed benign-looking but malicious knowledge
  - **Impersonation & masquerading** → appear as trusted users, tools, or roles
  - **History manipulation & delayed execution** → hide intent over time

ATLAS Tactics: (8) Defense Evasion - Examples

- **Prompt obfuscation**
  - Split intent across turns or encode it (base64, math, pseudo-code)
  - Ask the model to *transform* rather than act
  - Meaning survives, validators don't ☺️

- **Core idea**
  - Filters operate on **strings**
  - Attacker + model communicate via **representation changes**

- **Bypassing output filters**
  - Probe blocked words/topics iteratively
  - Reconstruct hidden content via synonyms, structure, or hints
  - Deterministic filters become **enumerable**

ATLAS Tactics: (9) Credential Access

- Adversaries obtain **valid credentials** by abusing AI systems, agents, or their surrounding infrastructure.

- **What happens** if we let an LLM access to a repository with committed secrets?
  - Files with secrets are indexed, just like code and documentation
  - API keys, tokens, passwords copied to RAG

  - The copilot is asked a *legitimate-sounding question*:
  - **“How do we authenticate to service X in prod?”**

- No jailbreak, No prompt injection, No bug… **What can we do here?**
  - **Data hygiene issue – LLMs make existing data more accessible**

ATLAS Tactics: (10) Discovery

- Adversaries map your AI environment, capabilities, and boundaries **to immediately enable exploitation**, not just reconnaissance for later steps.

- Discovery in AI systems often *is* the attack: **learning the system prompt**, tool definitions, or agent triggers can directly unlock privilege escalation, data access, or execution.
  - Some companies consider the system prompt to be one of biggest intellectual property secrets!

    - **For example – let's think about Claude Code**

ATLAS Tactics: (11) Lateral Movement

- Adversaries move through your AI environment by pivoting between **models, agents, data, and AI ops infrastructure**, expanding control beyond the initial access point.

- **How lateral movement works in AI systems**
  - Pivot from **AI apps → agents → underlying tools**
  - Move across **AI ops components**: model registries, vector DBs, notebooks, experiment trackers
  - Abuse shared credentials, service accounts, and cached tokens
  - Leverage **over-privileged AI agents** as bridges between systems

- In AI environments, lateral movement is often **API-to-API**, not host-to-host.

ATLAS Tactics: (12) Collection

- Adversaries gather **AI artifacts and high-value data** needed to achieve their objective or prepare for exfiltration and follow-on attacks.

- **What gets collected**
  - **Service data** from AI platforms, agents, and pipelines
  - **RAG databases**: indexed internal knowledge, documents, secrets
  - **Agent tool outputs**: API responses, logs, intermediate results

  - **AI artifacts**: models, weights, prompts, embeddings, configs
  - **Training & inference data** from repos, object stores, local disks

ATLAS Tactics: (13) Exfiltration

- **Core problem**
  - Exfiltration mostly happens through **normal AI usage**
  - Answers *are* the data channel

- **What attackers do**
  - Ask many small, valid questions
  - Stay below rate and volume thresholds
  - Turn models into **data compressors**

- **All these stay below detection threshold, as traffic seems normal**

ATLAS Tactics: (13) Exfiltration – examples

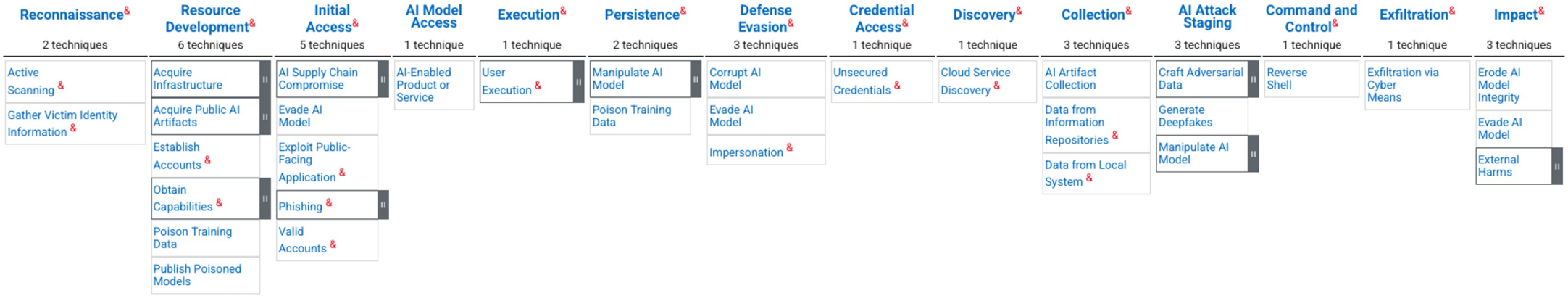
- **Inference API abuse**
  - **Model extraction** via repeated queries
  - **Membership inference** reveals training data
- **Prompt & system leakage**
  - System prompts extracted via indirect queries
  - Guardrails revealed through probing
- **Agents as exfil tools**
  - Agents pull data from multiple services
  - Results exfiltrated via “helpful summaries”
- **Example**
  - Copilot or RAG system summarizing internal docs → sensitive data leaves as text

ATLAS Tactics: (14) Impact

- Adversaries cause damage by disrupting AI availability, corrupting AI behavior, or exploiting AI-driven business processes.

- **Impact categories**
  - **Technical:**
    - Denial of service, Cost harvesting / token drain or Spamming
  - **Integrity:**
    - Poisoned outputs, Silent misclassification, Decision drift over time
  - **Business / Process:**
    - Manipulated workflows (approvals, triage, automation)
    - Fraud, Compliance and policy violations
  - **External harm:**
    - Financial loss, Reputational damage
  - **Intellectual Property:**
    - Model extraction, Training or sensitive data leakage
    - Prompt, pipeline, or system-design theft

ATLAS Matrix: Realized techniques



Questions?