Casey Fan, Diyana Tial
Mentors: Dianxiang Xu, Vladislav Dubrovenski

Secure LLM Agents, Code Vulnerabilities

Problem Statement

This project investigates how Large Language Model (LLM) agents can be exploited to violate key cybersecurity principles such as the CIA triad: confidentiality, integrity, and availability - through improper tool use, unsafe code generation, and agent coordination flaws. As LLM agents grow in complexity and autonomy, combining tool execution, planning, and collaboration within different agents, their attack surface expands significantly.
We aim to identify and evaluate vulnerabilities in agents' behavior and architecture, including but not limited to attacks such as adversarial prompt injection, insecure tool invocation chains, and unsafe code outputs. Exploring different AI frameworks such as AutoGen and LangChain and different online open-source projects, we will test these agents against known threats and hope to uncover new threats.

Literature Review

A Comprehensive Survey in LLM(-Agent) Full Stack Safety: Data, Training, & Deployment

Wang, Kun, Guibin Zhang, Zhenhong Zhou, Jiahao Wu, Miao Yu, Shiqian Zhao, Chenlong Yin et al. "A comprehensive survey in llm (-agent) full stack safety: Data, training and deployment." *arXiv preprint arXiv:2504.15585* (2025).

Although we first reviewed this paper a week ago, we continue to draw on it in our research because it offers a comprehensive survey of large language models, LLM-agents, and multi-agent systems–covering their vulnerabilities and security gaps, existing defense strategies, and evaluation metrics–which greatly enriches our understanding.
For this week, we decided to reread and focus on Chapter 6 of the paper, specifically 6.2 Single-Agent Safety and 6.3 Multi-Agent Safety since these sections explore the attacks of single-agent systems and multi-agent systems.
For 6.2 Single-Agent Safety, the authors provided many exploits that an attacker could potentially implement. These are:

1. **Injection**: There are two types of injections–prompt injection and tool injection. Prompt injections are malicious instructions that are embedded in input data whereas tool injections are malicious tools injected to enable further exploitation, such as using the tool to execute malicious actions.

2. **Backdoor**: A hidden trigger or malicious shortcut planted inside the agent so that when it sees a specific input, it behaves in a way the attacker wants and is often done in secret.
3. **Manipulation**: Directly or indirectly manipulating or altering the tool's returned content to leak sensitive information or carry out malicious actions.

The authors also dive into the memory aspect of exploitations such as:

1. **Memory poisoning**: These are adversarial attacks where malicious data is injected into an agent's long term memory.
2. **Privacy leakage**: Attackers exploit the interface between an agent and its long term memory to extract stored sensitive data
3. **Memory Misuse**: This is where an attacker plays a long game–using a sequence of questions that build on what the agent "remembers" in the current session to slowly wear down its safety checks. Another strategy of this exploit is when an attacker ladders up through multiple prompts since the agent only has short term memory.

For 6.3 Multi-Agent Safety, the types of exploits that were written were:

1. **Transmissive Attack**: A virus that jumps from one agent to the next in a MAS, silently infecting them all.
2. **Interference Attack**: This attack focuses on how it interferes with and disrupts interactions within the MAS, emphasizing communication disruption and misinformation. There are three types of attacks that does this which are:
    a. Jamming the channel
    b. Man-in-the-middle
    c. Misinformation Flooding
3. **Strategic Attack**: Involves collaboration between agents and strategic optimization of attack methods such as persuasive injection and knowledge manipulation injection.

Our research is built upon what this paper writes about in these two sections because we can use these different attacks/exploits and try them in the open source programs from GitHub.

Secure Design Principles

Xu, Dianxiang. "Secure Design Principles." Presentation, Software Security, Kansas City, MO.

This presentation constructed by our mentor Dr. Dianxing Xu includes the secure design principles. There are 10 contents on this topic which are: Minimal Attack Surface, Least Privilege, Defense-In-Depth, Fail-Safe Stance, Secure by Default, Separation of Duties, Avoidance of Security Through Obscurity, Robust Resource Management, Forensic Readiness, and Security Features ≠ Security. This presentation explains that secure design should consider preventive, detective, and protective measures to impede the progress of an intrusion and reduce its consequences. By understanding the security aspect of the design of the program, it'll be more helpful in giving us ideas on what exploits to break these secure barriers.

Threat Modeling

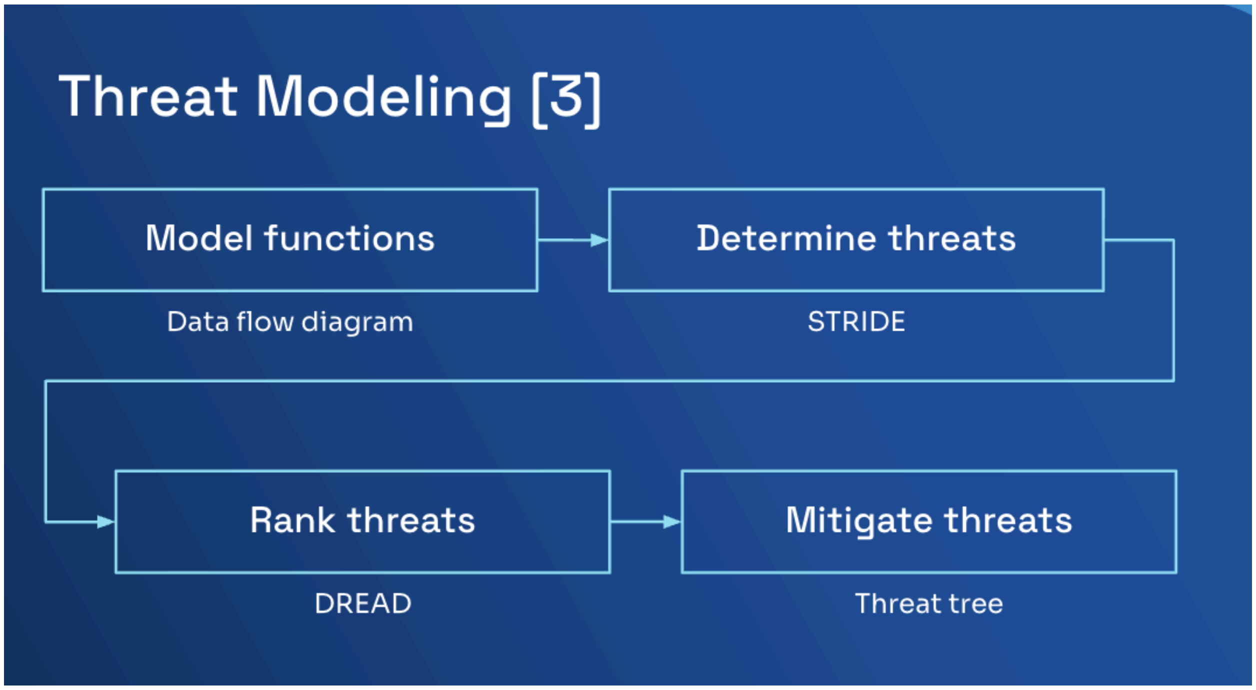
Xu, Dianxiang. "Threat Modeling." Presentation, CS 483/5583 Software Security, Kansas City, MO.

This presentation constructed by our mentor Dr. Dianxiang Xu goes in depth on threat modeling which is the process of identifying, specifying, evaluating, and mitigating potential attacks. In threat modeling, there are four sections that outlines this process and these are:

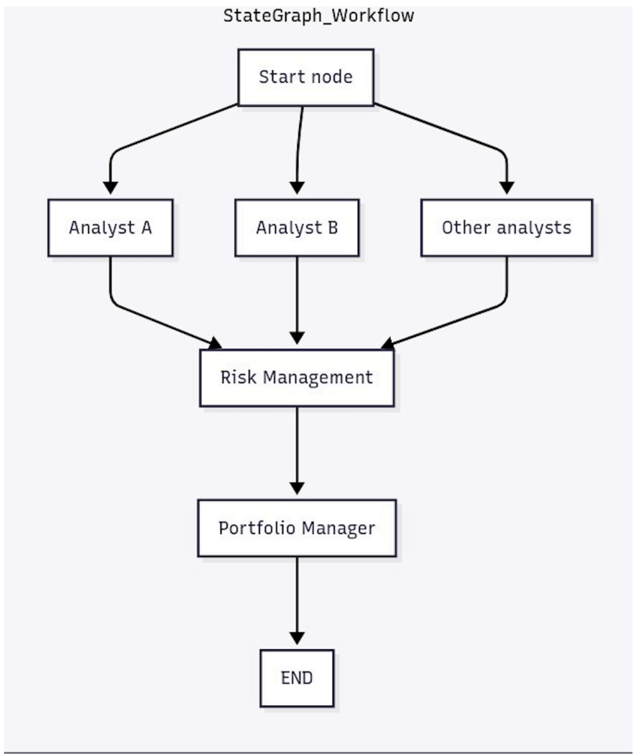
1. Model functions
2. Determine threats
3. Rank threats
4. Mitigate threats

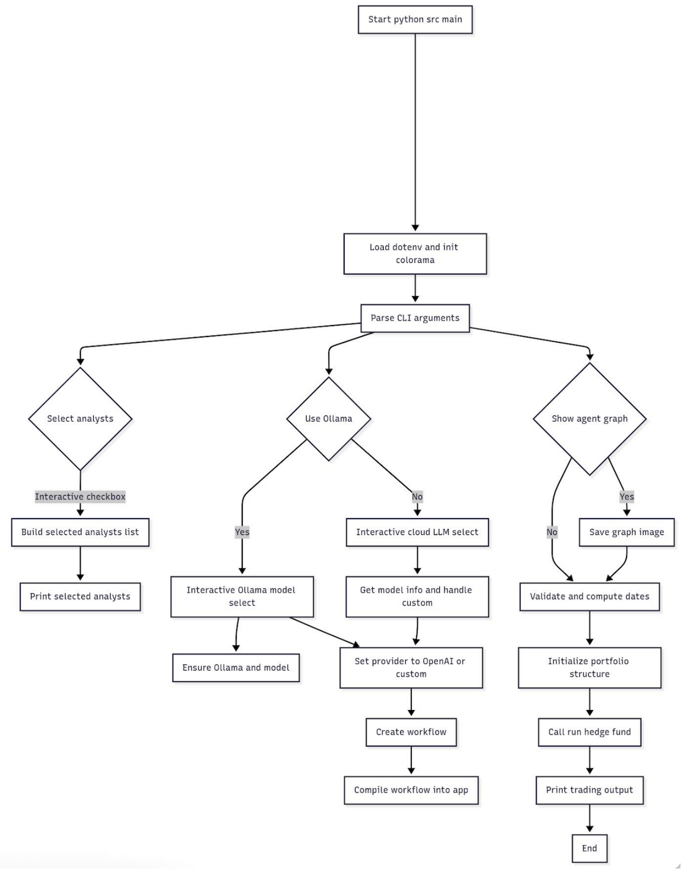
We mainly focus on 1. Model functions and 2. Determine Threats for our part of the research. With threat modeling, we are able to understand our application/what we are working with better. In week 4, we chose a highly starred open-source GitHub project (AI Hedge Fund) to attack, hoping its popularity would make it an interesting target. We tried several exploit techniques such as prompt injection and function enumeration but we quickly hit a wall–the program's strict input validation and scattered codebase prevented our efforts. We looked back and realized we should have spent time reading through its files and understanding the design of the program before jumping straight into attack implementation. This presentation gives us more understanding on why modeling your application is important before doing any experiment on it because how can you work with something you are not familiar with? Henceforward, we will take this lesson we learned and apply these techniques from this presentation for any other project we decide to use next.

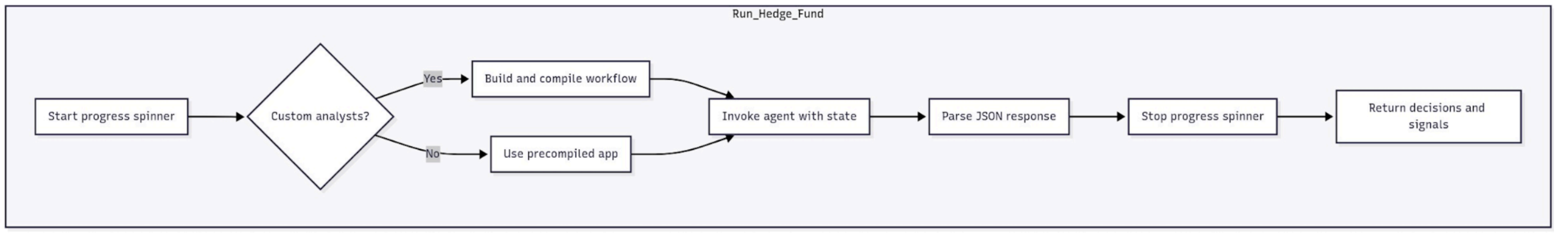
Conceptual Diagram



AI Hedge Fund Program Flow Diagram:




Start python src main
Load dotenv and init colorama
Parse CLI arguments
Select analysts
Use Ollama
Show agent graph
Interactive checkbox
Build selected analysts list
Print selected analysts
Yes
No
Interactive cloud LLM select
Interactive Ollama model select
Get model info and handle custom
Ensure Ollama and model
Set provider to OpenAI or custom
Create workflow
Compile workflow into app
No
Yes
Save graph image
Validate and compute dates
Initialize portfolio structure
Call run hedge fund
Print trading output
End



Next Steps

Casey

I plan on focusing on MAS interactions between agents mainly, but I also have an idea for an attack against the AI-hedge-fund program that isn't focused as much on agent communication. For MAS, I want to further research how agents interact with one another and how they "remember" their interactions with each other. I want to see if I can introduce an attack that involves a 'disruptive but cooperative' agent, and investigate whether that impairs performance or not. In a debating scenario, for example, this agent would provide factually correct information and sound analysis of other agents' arguments, but would also be more 'dramatic' or 'gossipy,' badmouthing other agents. My goal is to see if I can establish some distrust/tension between agents that leads to poorer communication and cooperation, making performance worse over a longer period of time. I believe that this attack can be especially useful in MASs that use RAG to have agents remember their interactions with each other, which could be the case for larger projects (eg, software development) or 'personable' MASs. This attack will be harder to detect than previous attacks that make themselves known immediately: developers might think they have a bug in their system instead of suspecting that one of the agents is responsible for the worse performance.
As for the attack on the AI-hedge-fund program, I noticed that users are allowed to put in their own custom model to run the program. My idea was to fine-tune a model to create a backdoor, so that when the model detects certain tickers, it tends to tell the user to BUY or SELL more often. An example of this would be if Google wanted users to buy more of their stock, they could create/fine-tune a model that is advertised as 'tuned for finance,' but this model also has GOOG as a trigger to always result in BUY, and AAPL as a trigger to always result in SELL/HOLD.

Diyana

I plan on focusing on creating an agent in the middle attack for the AI Hedge Fund program.
To do this, I would need to intercept the communication between agents by inserting a malicious intermediary agent. This agent would monitor, manipulate, or alter the "results" of the hedge fund program. With this program, it has a chain style communication where analysts are chosen, then it goes to the risk manager and finally, the portfolio manager takes action by deciding to buy, cover, sell, short, or hold. I plan to insert the malicious agent between either the analysts and risk manager or in between the risk manager and portfolio manager.
I am also planning on using tool injection as an attack for the AI Hedge Fund program. I believe that if I can somehow manipulate the one or more of the tools that these agents have access to, then I can successfully exploit this program. However, I don't really have an outline of what I want to do for this attack yet.
Lastly, I will also focus on another open source program called Agentcy. Agentcy is a multi-agent creative collaboration team of autonomous agents that takes two simple inputs to generate a plan, research and provide advice to help the user be successful in a business idea they have. For this program, it will mostly be all prompt injections since the input is not as restricted. I want to do more DoS (denial of service) attacks where the agent gets very confused from the

users’ input and just breaks down completely. Along with that, I want to try and see if I can manipulate it to accept “harmful” business ideas that is in a way breaking the law.