Harnessing LLMs for AppSec

Seth Law & Ken Johnson

github.com/absoluteappsec/harnessing_llms

wifireg.defcon.org

Introductions - Seth Law

Founder of **Redpoint Security**

Based in Salt Lake City, Utah

Consultant, Programmer, Trainer, Speaker

@sethlaw

Introductions - Ken Johnson

CTO DryRun Security // BJJ Nerd

Building AI powered AppSec Automation

AppSec Background:

- GitHub
- CTO @ “a consultancy”
- LivingSocial
- FishNet Security (now Optiv)
- US Pentagon
- Ex-Navy

Introductions

- Name or handle
- Current job/title/interest/career path
- Favorite LLM (or programming language)

Course Overview

Day 1:

- Intros & Overview
- Lab Setup
- LangChain Overview
- Prompts / Engineering
- Context
- Embeddings & Vector Stores
- LLM Exploration
- ChatBot Assistant
- Dynamic AppSec AI

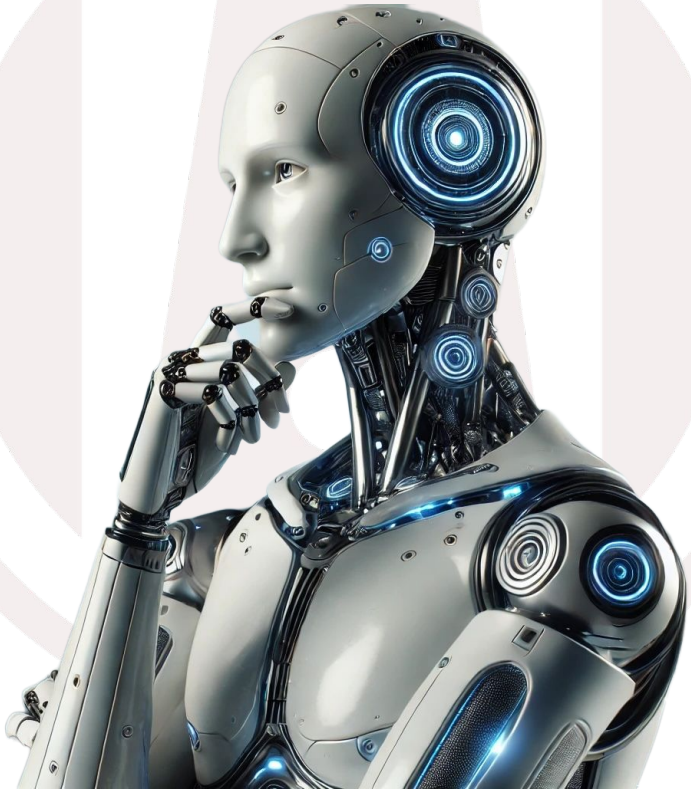
Day 2:

- Source Code Analysis
- Agentic AI
- Logging & Monitoring
- Testing
- Group Projects

Background behind the Course

- We are using this technology today and seeing massive benefits
  - Explanations of Code Changes - Pull Request Summaries
  - Helps easily categorize changes in an organization like new endpoints, authentication mechanisms, new infrastructure, etc.
  - Deconstructing code bases and providing compositional analysis
  - Detection of anomalous patterns
  - Creating risk scores and applying them to various security related analysis
  - Chat - extending your team by answering first level questions
  - … really just uncovering new use cases all the time
- We want the Security industry to embrace this new technology! Not get left behind

Gen AI Concepts - How do LLMs work?



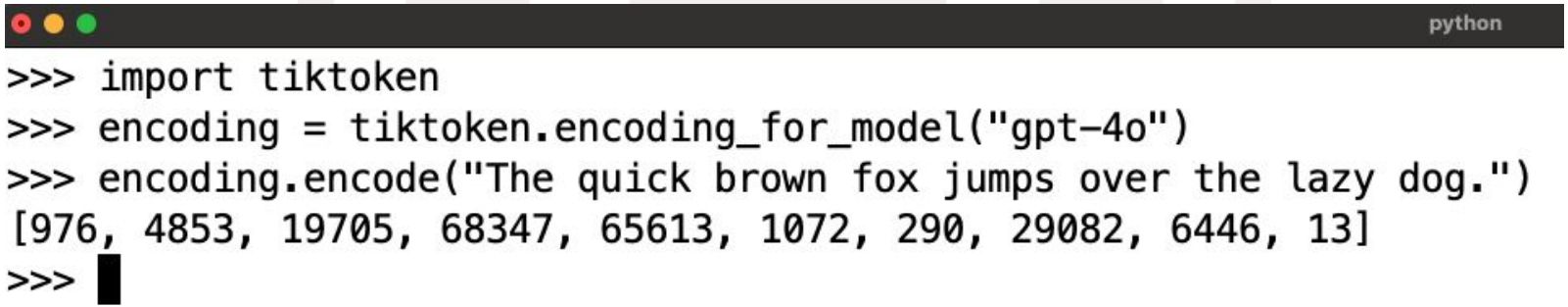
Gen AI Concepts - How do LLMs work?

Large Language Models / Small Language Models

- How do they work
  - Tokens & Prediction
  - Inference
  - Prompting
  - Training vs RAG
- Common Misconceptions
- Notable events

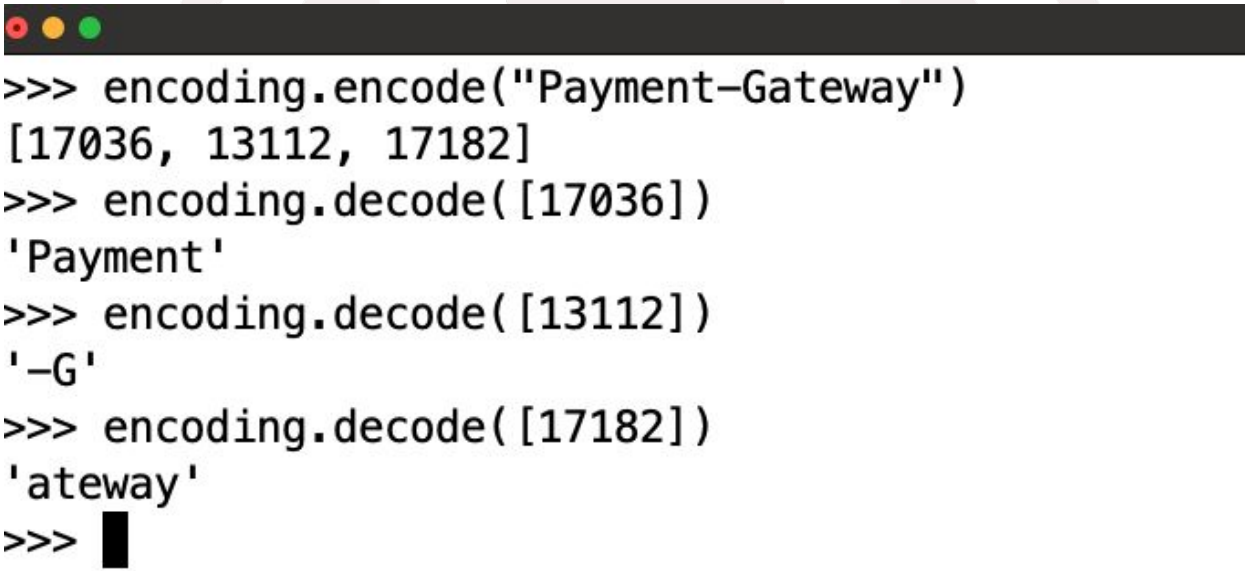
Gen AI Concepts - How do LLMs work?

BPE - Byte Pair Encoding creates a vocabulary for the LLM mapping characters to tokens



Gen AI Concepts - How do LLMs work?

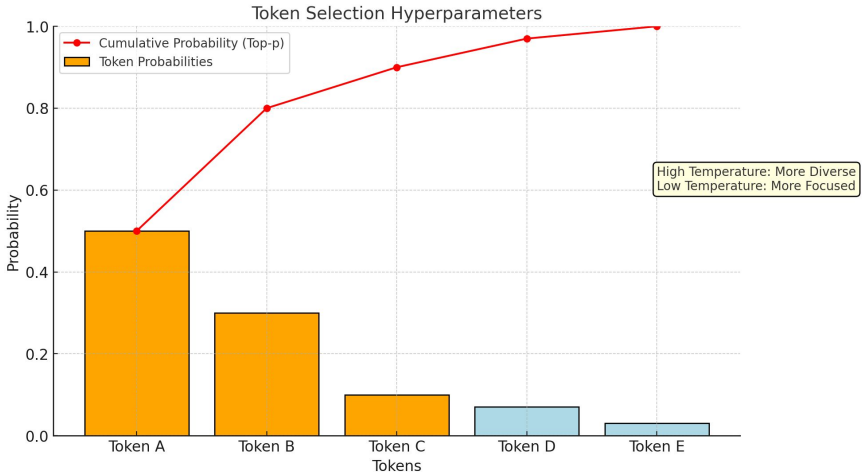
Tokens & Token Prediction



Gen AI Concepts - How do LLMs work?

Token Prediction in LLMs:

1. **Input**: Receive tokenized inputs from the user.
2. **Prediction**: Generate a ranked list of the most probable next tokens based on the input sequence.
3. **Selection**: Choose the next token using either:
    - **Greedy**: Select the highest-probability token.
    - **Probabilistic**: Random selection weighted by the assigned probabilities.
4. **Control**: Probabilities are influenced by hyperparameters like:
    - **Temperature**: Adjusts randomness (higher = more random).
    - **Top-k**: Limits choices to the top-k highest probabilities.
    - **Top-p (nucleus sampling)**: Limits choices to a cumulative probability threshold.



Gen AI Concepts - How do LLMs work?

Hyperparameters for Token Selection

1. **Temperature**
   - Adjusts randomness in token selection.
   - **Low values (e.g., 0.2):** Focused on high-probability tokens (deterministic).
   - **High values (e.g., 1.5):** More diverse and creative outputs (random).
2. **Top-p (Nucleus Sampling)**
   - Chooses from tokens that cumulatively cover a probability threshold (e.g., 90%).
   - Ensures only the most relevant tokens (based on context) are considered.
3. **Top-k**
   - Restricts token choices to the **k most probable options**.
   - Example: Top-3 only considers the three tokens with the highest probabilities.

Gen AI Concepts - How do LLMs work?

Helpful Articles:

https://writings.stephenwolfram.com/2023/02/what-is-chatgpt-doing-and-why-does-it-work/

https://blog.miguelgrinberg.com/post/how-llms-work-explained-without-math

Gen AI Concepts - How do LLMs work?

Training:

- I love AppSec
- You love AppSec
- We love AppSec

-	I	you	we	love	AppSec
I				1	
you				1	
we				1	
love					3
AppSec					

Gen AI Concepts - How do LLMs work?

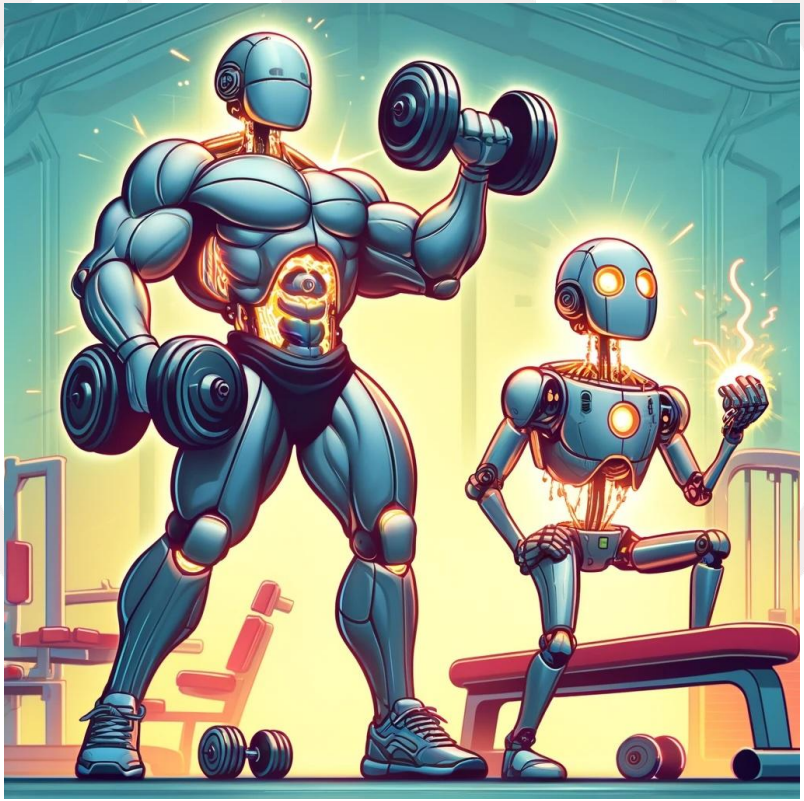
Training:

- I love AppSec
- You love AppSec
- We love AppSec

-	I	you	we	love	AppSec
I				100%	
you				100%	
we				100%	
love	33.3%	33.3%	33.3%		100%
*AppSec	**25%**	**25%**	**25%**	**25%**	

* **Denotes a token that has no “next” tokens (ie - “Hole”)**

LLMs Strengths & Weaknesses



LLMs Strengths & Weaknesses

Strengths

1. Summarization of Code Changes
2. Identifying Intent and Behavior
3. Pattern Detection and Prediction
4. Automating Repetitive Tasks
5. Contextual Awareness
6. Support for Multiple Languages and Frameworks
7. Efficiency and Scalability

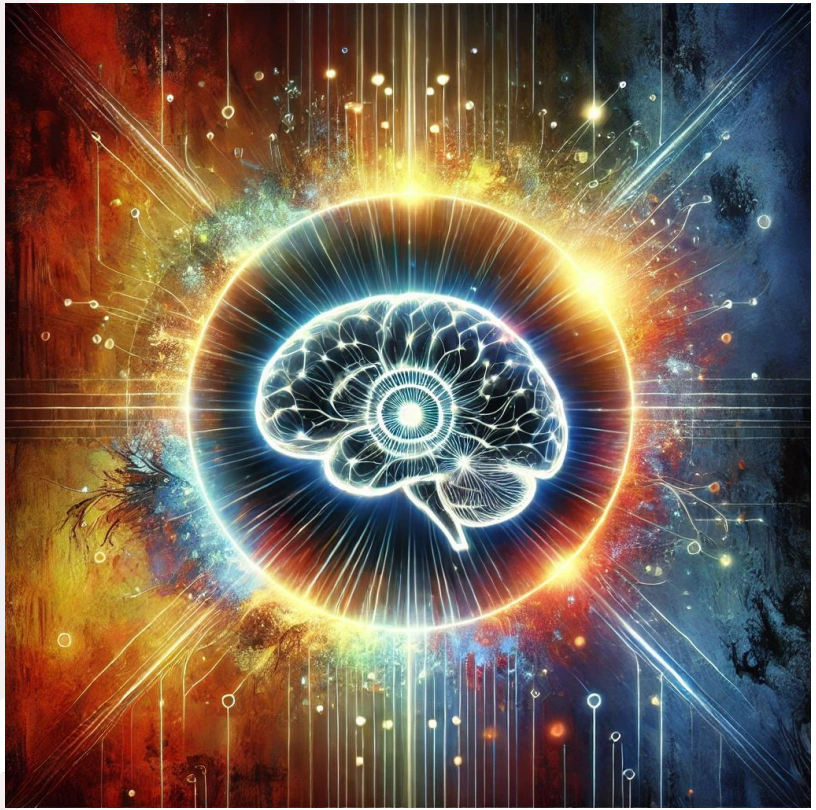
Weaknesses

1. Non-Deterministic Outputs
2. Hallucination of Risks
3. Resource Intensity
4. Lack of True Causal Reasoning
5. Dependency on Training Data
6. Difficulty Handling Complex Codebases
7. False Confidence in Results
8. Ambiguity in Prompts

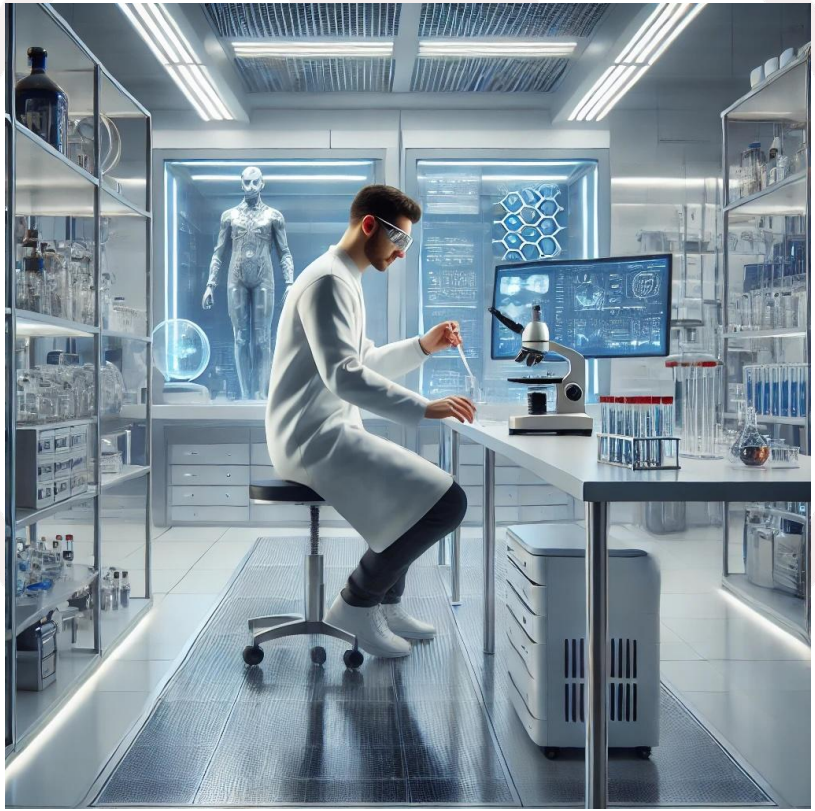
LLMs Strengths & Weaknesses

We can mitigate almost all LLM deficiencies via:

- Context
- Prompt Engineering
- Technique
- Tuning
- Agents & Custom Tools
- Proper Testing



Lab Setup



Lab Setup

1. Clone the following repo: https://github.com/absoluteappsec/harnessing_llms
2. Navigate to cloned repo, type:

```
$ python -m venv venv

$ source venv/bin/activate

$ pip install -r requirements.txt
```

3. Download your .env file (Gist link provided to you in an email) and place in the root of the cloned repo

LangChain

LangChain

Simplifies process of creating advanced AI applications

Provides abstractions for many things such as:

1. Chains - Combines multiple utilities into a single workflow
2. Memory - Adds state to chains, retaining context across multiple interactions
3. Agents - Enable dynamic decision-making, where the system determines what actions to take
4. Retrieval Augmented Generation (RAG) - Integrate with external knowledge bases to enhance outcomes
5. Much more!

LangChain - Equivalent Frameworks

Haystack - https://haystack.deepset.ai/

Semantic Kernel - https://github.com/microsoft/semantic-kernel

LLamaIndex

https://github.com/run-llama/llama_index

Comparison Table

Feature	LangChain	LlamaIndex	Haystack	Semantic Kernel	Hugging Face Transformers	OpenAI Function Calling
Workflow Management	✅	Limited	✅	✅	❌	❌
Memory Support	✅	❌	✅	✅	❌	❌
RAG Support	✅	✅	✅	✅	❌	❌
Indexing	✅	✅	✅	❌	❌	❌
Ease of Use	High	Moderate	Moderate	High	Low	High

LangChain

LangChain has 3 primary products:

- LangChain (OSS) - https://github.com/langchain-ai/langchain
- LangSmith (Paid) - https://smith.langchain.com/
- LangGraph (Paid) - https://www.langchain.com/langgraph

LangChain

Things to know before getting started:

- Frequent changes means even minor version changes can break production systems

- Documentation https://python.langchain.com/docs/introduction

Primary Components:

langchain-core: Base abstractions and LangChain Expression Language (LCEL).

Langchain: Chains, agents, and retrieval strategies

Langchain-community: Third party integrations, community-maintained

LangChain - Navigating Docs

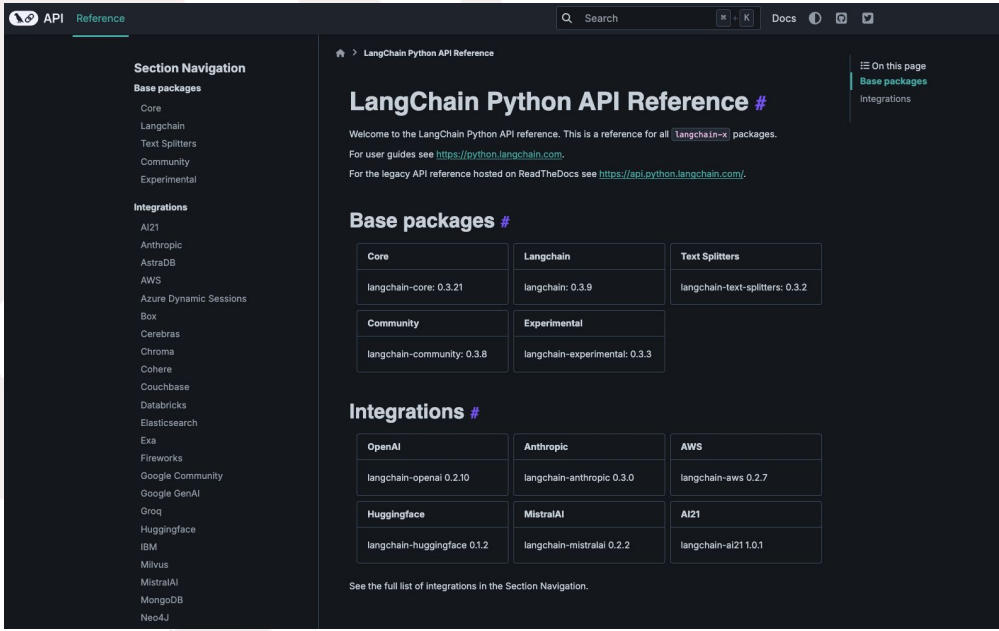
Introduction: Good way to get a lay of the land

Tutorials: Dig in right away to some examples of building different AI applications

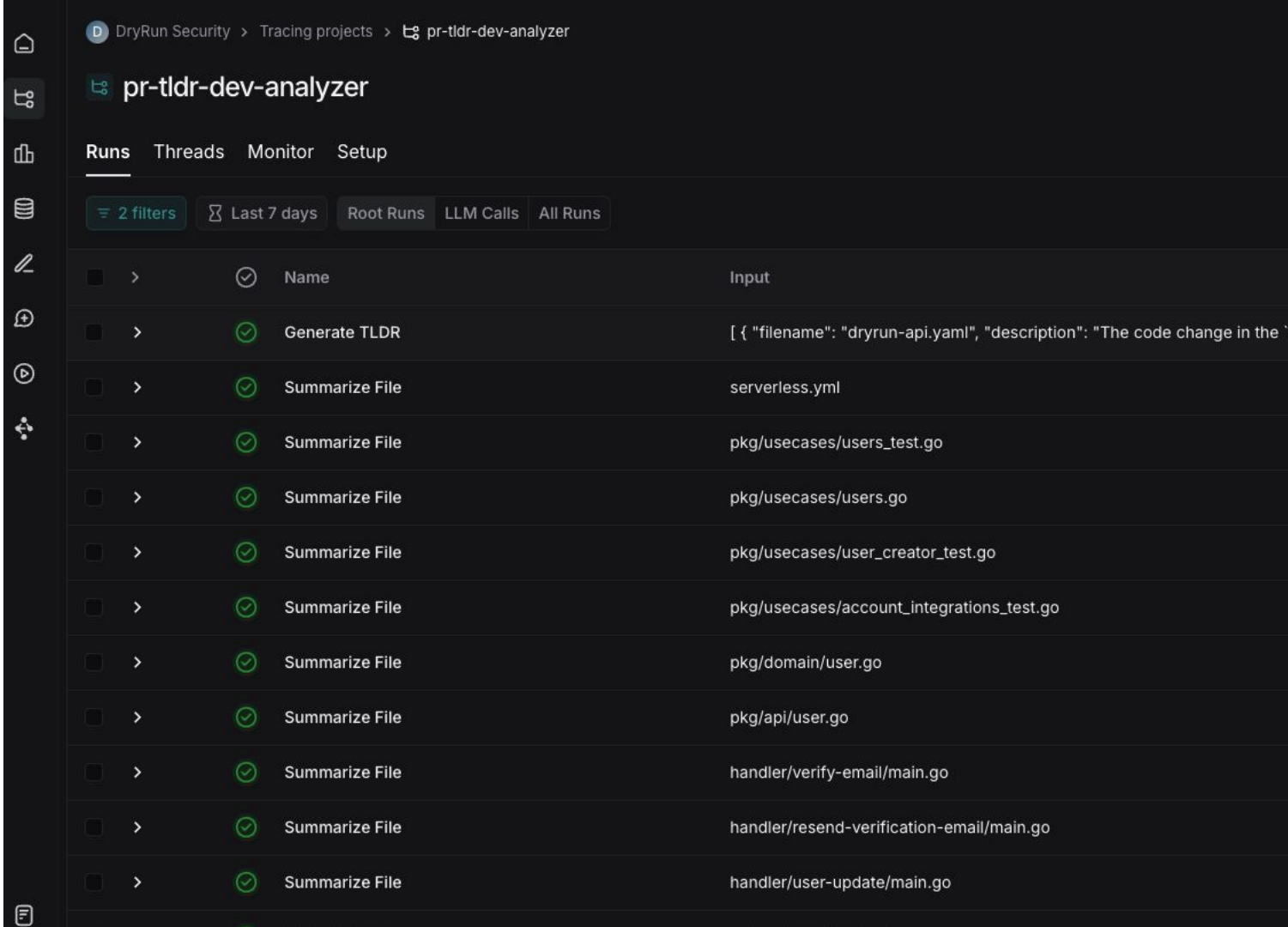
How-to Guides: Guides that are goal oriented/concrete

Conceptual Guide: Provides some high level information on the various components that you will utilize within LangChain

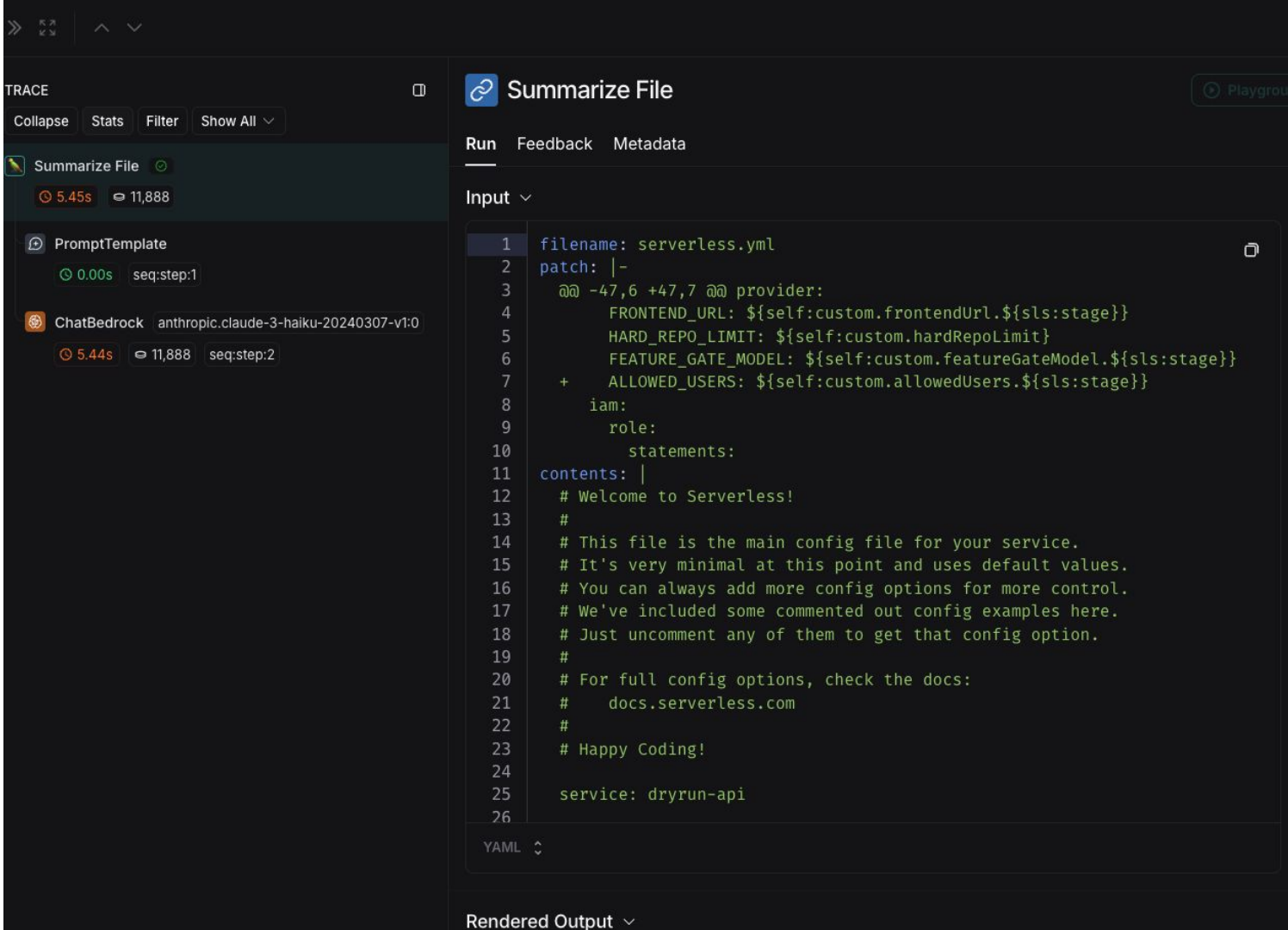
API Reference:



LangSmith


DryRun Security > Tracing projects > pr-tldr-dev-analyzer
pr-tldr-dev-analyzer
Runs
Threads
Monitor
Setup
2 filters
Last 7 days
Root Runs
LLM Calls
All Runs
Name
Input
Generate TLDR
[{ "filename": "dryrun-api.yaml", "description": "The code change in the
Summarize File
serverless.yml
Summarize File
pkg/usecases/users_test.go
Summarize File
pkg/usecases/users.go
Summarize File
pkg/usecases/user_creator_test.go
Summarize File
pkg/usecases/account_integrations_test.go
Summarize File
pkg/domain/user.go
Summarize File
pkg/api/user.go
Summarize File
handler/verify-email/main.go
Summarize File
handler/resend-verification-email/main.go
Summarize File
handler/user-update/main.go

LangSmith


TRACE
Collapse
Stats
Filter
Show All
Summarize File
5.45s
11,888
PromptTemplate
0.00s
seq:step:1
ChatBedrock
anthropic.claude-3-haiku-20240307-v1:0
5.44s
11,888
seq:step:2
Summarize File
Run
Feedback
Metadata
Input
filename: serverless.yml
patch: |-
@@ -47,6 +47,7 @@ provider:
FRONTEND_URL: ${self:custom.frontendUrl.${sls:stage}}
HARD_REPO_LIMIT: ${self:custom.hardRepoLimit}
FEATURE_GATE_MODEL: ${self:custom.featureGateModel.${sls:stage}}
+ ALLOWED_USERS: ${self:custom.allowedUsers.${sls:stage}}
iam:
role:
statements:
contents: |
Welcome to Serverless!
#
This file is the main config file for your service.
It's very minimal at this point and uses default values.
You can always add more config options for more control.
We've included some commented out config examples here.
Just uncomment any of them to get that config option.
#
For full config options, check the docs:
docs.serverless.com
#
Happy Coding!
service: dryrun-api
YAML
Rendered Output

LangGraph

Keeps agent workflows resilient: tasks can pause and resume without losing progress

Handles throttling & errors gracefully: no need to restart from scratch

Uses checkpoints to manage state: supports Memory, SQLite, Postgres, MongoDB, and Redis backends for persistence and recovery

```python
from langgraph.graph import StateGraph
from langgraph.checkpoint.redis import RedisSaver

# 1. Define workflow steps
def step1(state):
    return {"value": state["value"] + 1}

def step2(state):
    return {"value": state["value"] * 2}

# 2. Build the state graph
graph = StateGraph({"value": int})
graph.add_node("start", step1)
graph.add_node("end", step2)
graph.set_entry_point("start")
graph.set_finish_point("end")

# 3. Add Redis checkpointing
checkpointer = RedisSaver(url="redis://localhost:6379")

# 4. Run with persistence
result = graph.compile(checkpointer=checkpointer).invoke({"value": 5})
print(result)  # {'value': 12}
```

LangGraph (Agentic development)

https://langchain-ai.github.io/langgraph/

Server: Opinionated API / Arch for building agentic apps

Studio: IDE that connects to a server for visualization/debugging

CLI: Interface that enables local interactions with LangGraph

SDK: Provides a programmatic way to interact with deployed LangGraph Applications

Remote Graph: Allows you to interact with any deployed LangGraph application as though it were running locally

Prompt Engineering

Prompt Engineering

Prompts are the input instructions or text provided to a large language model (LLM) to guide its response. They set the context, define the task, and influence the quality and relevance of the output.

💡 Be mindful of context length windows however… they're fairly large these days

📚Helpful Guide
https://github.com/dair-ai/Prompt-Engineering-Guide

Context: Background information on task

Instruction: Specify what the LLM should do

Input Data: Dynamic/user input and usually defined as a Human or User prompt

Output Format: Defines how the response should be structured

Constraints or Rules: If you need to define how the response should look

Examples: Usually done with few shot prompting

Tone/Style: Describes the style in which the LLM should response (ie - professional, pirate, yoda, etc.)

Prompt Engineering

Some popular frameworks:

CO-STAR: Context, Objective, Scale, Time, Actor, Resources

RACE: Role, Action, Context, Explanation

CARE: Context, Action, Result, Example

APE: Action, Purpose, Execution

RICE: Role, Input, Steps, Execution

Prompt Engineering - Reflexion

Three Key Stages:

1. Initial Response: Generate first answer to the prompt
2. Self-Reflection: Critically analyze the response for gaps, assumptions, and potential improvements
3. Refined Output: Produce an enhanced response incorporating insights from reflection

- Improves accuracy and completeness
- Reduces hallucinations via self-verification
- Helps identify blind spots in initial analysis
- Produces more thoughtful and nuanced outputs

Prompt Engineering - Co-Star Example

Context: This code is part of a login system for a financial application.

Objective: Identify potential authentication security vulnerabilities in this code.

Scale: Focus only on the `authenticate_user` function

Time: This analysis must be completed in under 1 minute

Actor: You are an application security engineer.

Resources: Refer to OWASP's top 10 vulnerabilities for guidance.

Example - Reflexion

1. Initial Analysis:
    - First observations
    - Key findings
    - Initial conclusions
2. Reflection Phase:
    - What might I have missed?
    - Are my assumptions valid?
    - What other perspectives should I consider?
3. Final Enhanced Response:
    - Refined conclusions
    - Additional insights
    - More complete analysis

Exercise:
Craft Prompt

Exercise: Craft Prompt

🎯**Objective: Find out as much information about the Juice Shop application as possible**

Instructions:

1. Open `scripts/prompt_engineering.py`
2. Experiment with both system and user prompts to improve results.
3. Have fun! This is free form, use one of the prompt engineering frameworks to guide you
4. 🛑 **Only modify user questions and system prompts for now**

```
user_question = """Tell me the following information about the code base I am
providing you:
    - Purpose of the application
    - Web technologies used in the application
    - Templating language used in the application
    - Database used in the application
    - Authentication mechanisms used in the application
    - Authorization mechanisms used in the application


   List libraries by their name, purpose, and version that are
   used in the vtm application for the following categories:
      - Security
      - Testing
      - Documentation
      - Build
      - Database
      - Authentication / Authorization
      - HTML Templating (ex: pug, handlebars)
      - CSS Frameworks (ex: bootstrap, tailwind)
      - widgets / UI components
   """
```

Exercise: Craft Prompt

Retrospective:

- What worked and what did not?
- What kind of experiments did we run?
- What are your thoughts on how modification of prompts affect the outcome?
- Anything else?

Context

Context

Context is the background information or situational details provided in a prompt to help the LLM understand the task environment and relevance.

💡 Context can be a number of things just remember it's included in the “context window” length calculation

Can be provided in a number of ways:

- Loaded from a static source such as a yaml file, database entry, or markdown file
- Chained using a vector store
- Statically defined inside of the prompt

Context Examples

- Technical documentation
- Tribal Knowledge
- App components
- Call Graphs / Abstract Syntax Tree (AST)
- Source Code Files
- System or application logs
- Output of Scanner Findings

Context

Loading the contents of a markdown file to provide information about node.js web applications.

```
Typically, Node.js web application libraries will be located in a `yarn` file or
`package.json`

### When it comes to Node.js web applications, the following is a list of
security-specific libraries:

Input Sanitization/Validation
===

- Joi
- Validator
- Express-Validator

Authentication
===

- jsonwebtoken
- passport.js
- express-jwt

Authorization
===

- casbin
- accesscontrol

Secure Session Management
```

Contents of “juice_shop_knowldegebase.md”

Context

Now the code is “code” and the file contents are a “knowledgebase”

```
system_prompt_template = """
You are a highly skilled and detail-oriented code review assistant with expertise in
both application security and functional code analysis. Your role is to assist
developers and security professionals by providing accurate, concise, and actionable
insights.

Your task is to analyze the provided source code of a web application and answer
specific questions about its functionality, security, and technologies. Always
maintain a professional tone and prioritize clarity in your responses.

In your analysis:
- Clearly identify and explain the purpose and technologies used in the codebase.
- Highlight critical security mechanisms such as authentication and authorization.
- Provide details on libraries, tools, and frameworks, organized by their categories
and roles in the application.
- When relevant, make recommendations for improving security or functionality.

Search the following codebase to answer the questions:

<code>
{code}
</code>

Use the following context to help answer questions:
<context>
{context}
</context>
"""
```

```
chain = (
    {
        "code": retriever,
        "question": RunnablePassthrough(),
        "context": RunnablePassthrough()
    }
    | prompt
    | llm
    | StrOutputParser()
)
```

```
print('performing analysis....')
response = chain.invoke(input=user_question, context=context)
print(response)
```

We pass the context as “**`RunnablePassThrough()`**” and use **`.invoke()`** to pass in the context

Exercise:
Craft Context

Exercise: Craft Context

🎯Objective: Experiment with various types of context to improve the results of Juice Shop analysis

Instructions:

1. Open `scripts/building_with_context.py`
2. Open `data/juice_shop_knowledgebase.md`
3. Experiment with providing different types of context to improve the results of our LLMs analysis
4. **🛑 Only modify context and prompt**

Exercise: Craft Context

Retrospective:

- What types of context did you experiment with?
- Were there any combinations of prompt + context that worked better than others?

Embeddings & Vector Stores

Embeddings & Vector Stores

Embeddings:

Dense, numerical representations of data (e.g., words, sentences, or code) in a continuous vector space. They capture semantic meaning, enabling similar items (like related words or similar code snippets) to have closer representations in this space.

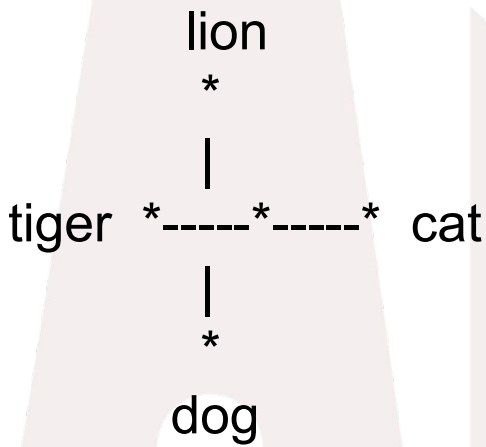
- Dense Representation - Fixed Size
- Semantic Encoding - Relationships between things
- Domain Specific - Can be pre-trained

Vector Stores:

Vector stores are specialized databases that store and manage embeddings. They enable efficient similarity searches by finding vectors close to a given query, often used in applications like document retrieval, recommendation systems, or question answering.

💡 For local development, we often use FAISS. For production purposes, Pinecone works very well. Many options are available!

Embedding Space - 2D grid for simplicity



- Points like "cat" and "dog" are close together because they are semantically related (both are common pets).
- "Lion" and "tiger" are also close, showing their similarity.
- Even though the grid is 2D here for visualization, actual embeddings are in a high-dimensional space.

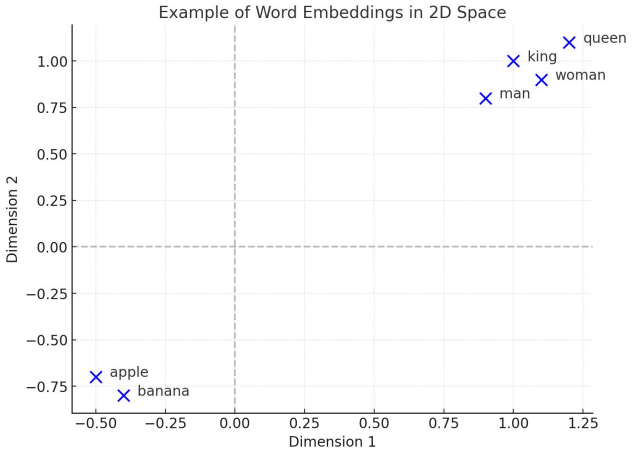
Embeddings & Vector Stores

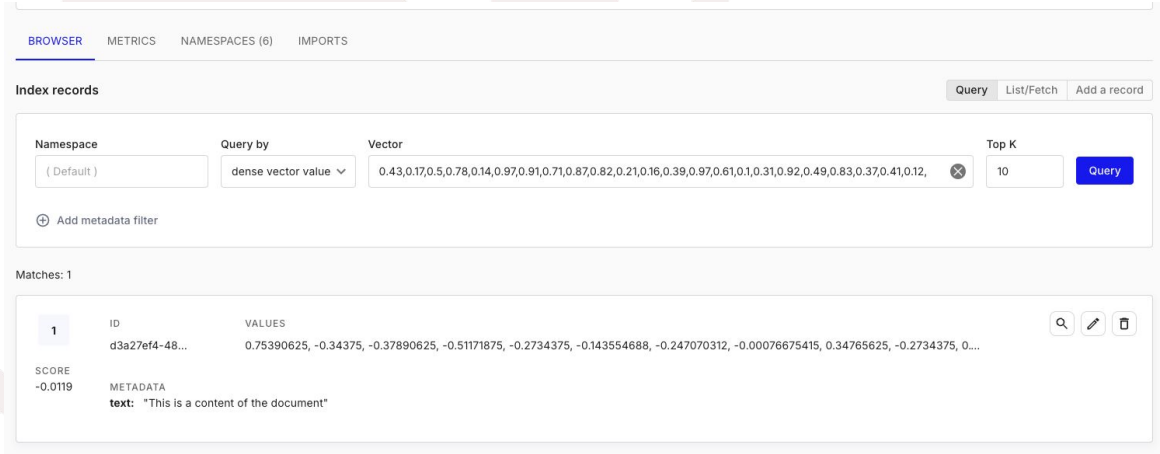
Embedding Format Example

For the word **"quick"**, a 3-dimensional embedding might look like this:

plaintext Copy code

```
[0.345, -0.221, 0.891]
```





Embeddings & Vector Stores

💡Not every LLM supports Embeddings, this is why you will notice that we utilize one LLM for inference tasks and another for performing Embeddings.

```
embeddings = BedrockEmbeddings(model_id='amazon.titan-embed-text-v1')

llm = ChatBedrock(
    model_id='anthropic.claude-3-haiku-20240307-v1:0',
    model_kwargs={"temperature": 0.6},
)
```

Embeddings & Vector Stores

Embeddings:

Comprehensive list of vector databases supported by LangChain
https://python.langchain.com/v0.1/docs/integrations/vectorstores/

For our local purposes, you'll notice that we're utilizing FAISS

💡Pro-Tip, save the index database so that you do not have to create embeddings and then store them every 👏 single 👏 time 👏

```
db = FAISS.from_documents(texts, embeddings)
db.save_local("juice_shop.faiss")
```

Embeddings & Vector Stores

Considerations:

- Embeddings & Vector stores must align dimension-wise
  - If the embedding LLM only supports 768 dimensions but you use a 1536 dimension db… issues will arise
- Cost
- Production or for local dev?
- How much control do you want?
  - Example: PGVector vs Pinecone

Embeddings & Vector Stores

Text Splitting:

- Size limitations - remember embedding is similar to an inference operation, it has a set context window length
- When splitting code, LangChain offers a utility which separates code by language
- To see the list of supported languages:

```
>>> from langchain_text_splitters import ( Language)
>>> [e.value for e in Language]
['cpp', 'go', 'java', 'kotlin', 'js', 'ts', 'php', 'proto', 'python'
, 'rst', 'ruby', 'rust', 'scala', 'swift', 'markdown', 'latex', 'htm
l', 'sol', 'csharp', 'cobol', 'c', 'lua', 'perl', 'haskell', 'elixir
', 'powershell']
>>>
```

```
loader = GenericLoader.from_filesystem(
    local_path,
    glob="**/*",
    suffixes=[".py"],
    parser=LanguageParser(language=Language.PYTHON),
    show_progress=True
)
```

```
splitter = RecursiveCharacterTextSplitter.from_language(language=Language.PYTHON,
    chunk_size=8000,
    chunk_overlap=20
)
texts = splitter.split_documents(documents)


db = FAISS.from_documents(texts, embeddings)
```

Exercise:
Embedding & Vectorstores

Exercise: Embed(dings) & (Vector) Store

🎯Objective: Load a repository of your choosing and store in our vector database FAISS, use previous code from "build_context.py" to analyze the database

1. Open **`scripts/embed_and_store.py`**
2. Modify the "RELEVANT" portions (specified in comments) to support the repo of your choice
3. Incorporate **`building_with_context.py`** code to build a feature complete script

Exercise: Embed(dings) & (Vector) Store

Retrospective:

- What type of repo(s) did you experiment with?
- What challenges did you face?
- Did you experiment with any of the splitting options?
- Are we feeling more comfortable with loading and analyzing code?

LLM Exploration

LLM Exploration

Considerations:

- Commercial vs OSS
- Transactional vs Dedicated
- Training vs Agentic / RAG (*can be all 3)
- Context-Window Size
- Purpose of the LLM

Commercial Vs OSS

Commercial

- Supports larger context windows
- Support usually is an option
- Good for transactional costs
- Bad option for training (though “fine tuning” options exist in some cases)

OSS

- Free to use, not to run
- No support
- Hosted on AI’s equivalent of “npm” (HuggingFace) so typical supply chain woes exist
- Low context window
- Best for training
- Fixed cost options exist

LLM Exploration

LLM	Context Window	Input Modalities	Accuracy for AppSec Tasks
GPT5	400k	Text/Image	High
Claude 3.7/4	200k	Text	High
Gemini 2.5 Pro	1 million	Text/Image	*High
DeepSeek R1	128k	Text	Low
LLaMa 3.3	128k	Text	Low
Mixtral/Mistral (large)	131k	Text	Moderate

* denotes not personally used but research performed

Chatbot Assistant

Chatbot Assistant

Common AppSec Use Cases:

1. Feedback mechanism for LLM analysis
2. Assistant on Slack to help answer first level triage questions
3. Answer coding related questions directly from GitHub/GitLab/etc.

Chatbot Assistant

Overview of Steps:

1. Load PDF into LangChain’s “Document” format
2. Create embeddings and store in vector database
3. Build Prompts
4. Chain components together and reference docs to answer questions

Chatbot Assistant

Supported Loaders: PDFs, Markdown, HTML, File Directories, JSON, and more…

Comprehensive list

- https://python.langchain.com/docs/integrations/document_loaders/

*You've actually already used a loader already 😀

Exercise:
Chatbot Assistant

Exercise: Chatbot Assistant

🎯**Objective: Load and save document, use as the source of knowledge for a chatbot, and answer questions while improving the existing prompt**

1. Use `for_instructors/load_guide.py` as inspiration and langchain "Document loaders" documentation for guidance
2. Create or download a document you would like to answer questions for
3. Store in `../vector_databases` utilizing a name of your choice
4. Utilize and modify `scripts/chatbot.py` to:
   a. Improving the prompt
   b. Load the vector database containing the document you wish to reference
5. Your chatbot, as much as is possible, should always respond with "Contact the security team" if the answer is not directly in the document

💯BONUS POINTS: 💯- Eliminate prompt injection

Exercise: Chatbot Assistant

Retrospective:

- What types of documents did you utilize?
- What challenges did you face trying to limit your LLM to referencing only what is in the document?
- What prompt techniques did you utilize to improve the outcome?
- Anything else to share?

*Did anyone try to create a chat history?

Dynamic AppSec AI

Dynamic AppSec AI - Lessons Learned

Window-size will be your enemy, splitting up dynamic traffic does not always lead to the desired result

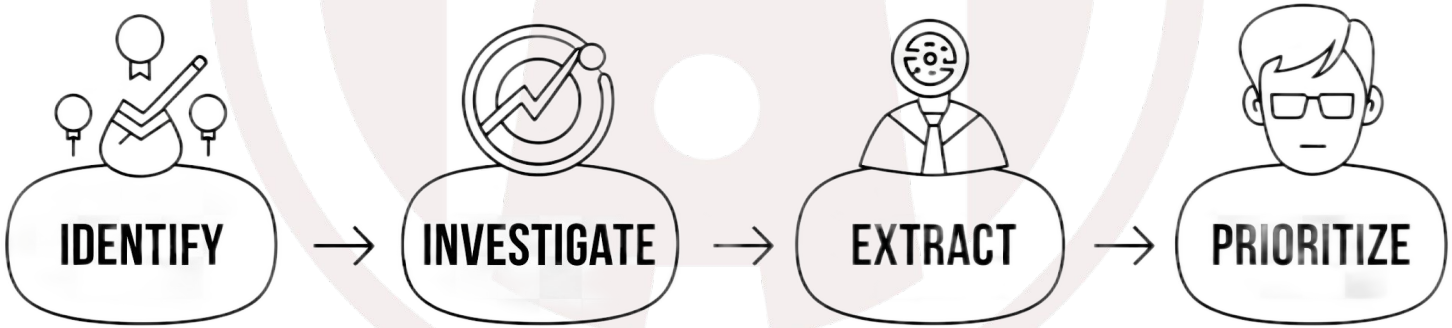
AI *is expected* to be helpful, which leads to hallucinations (in other words, false positives).

Start broad, use prompt engineering techniques and alternative AI to filter out noise.

Reflection and few-shot prompting act as a shortcut to building agents or MCP servers.

Dynamic AppSec AI - Recommendations

1. Limit analysis to specific vulnerabilities.
2. Use Reflection to filter out the noise during each interaction.
3. Build a pipeline of various prompts for each specific activity.



Dynamic AppSec AI - Human in the Loop

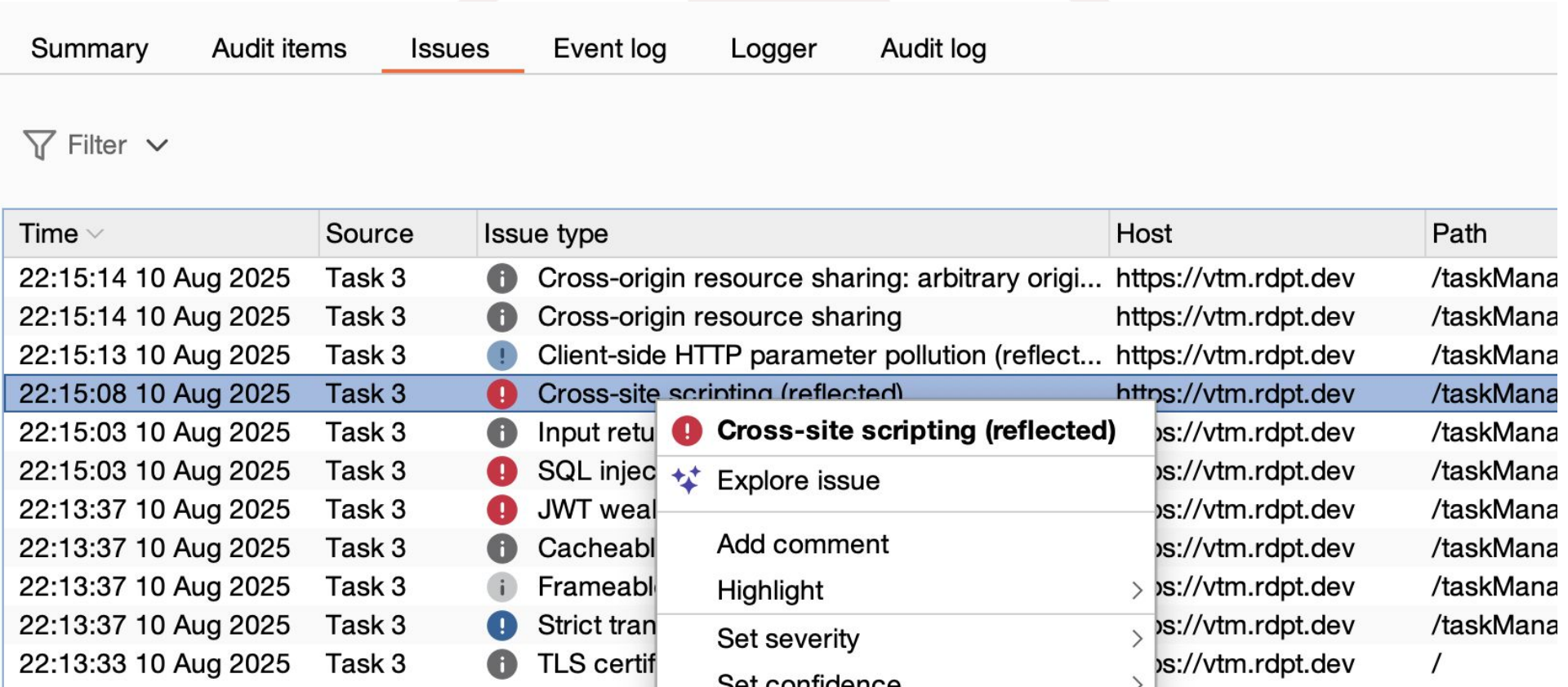
Dynamic AppSec AI - Burp AI vs. Custom

- Tool Native vs. AI Native
- AI Extensions for proxies/tools - Summarize discrete results, still building on fixed-pattern analysis
- Custom Tools/Workflows - Feels like a super assistant, give instructions of additional patterns, get good results

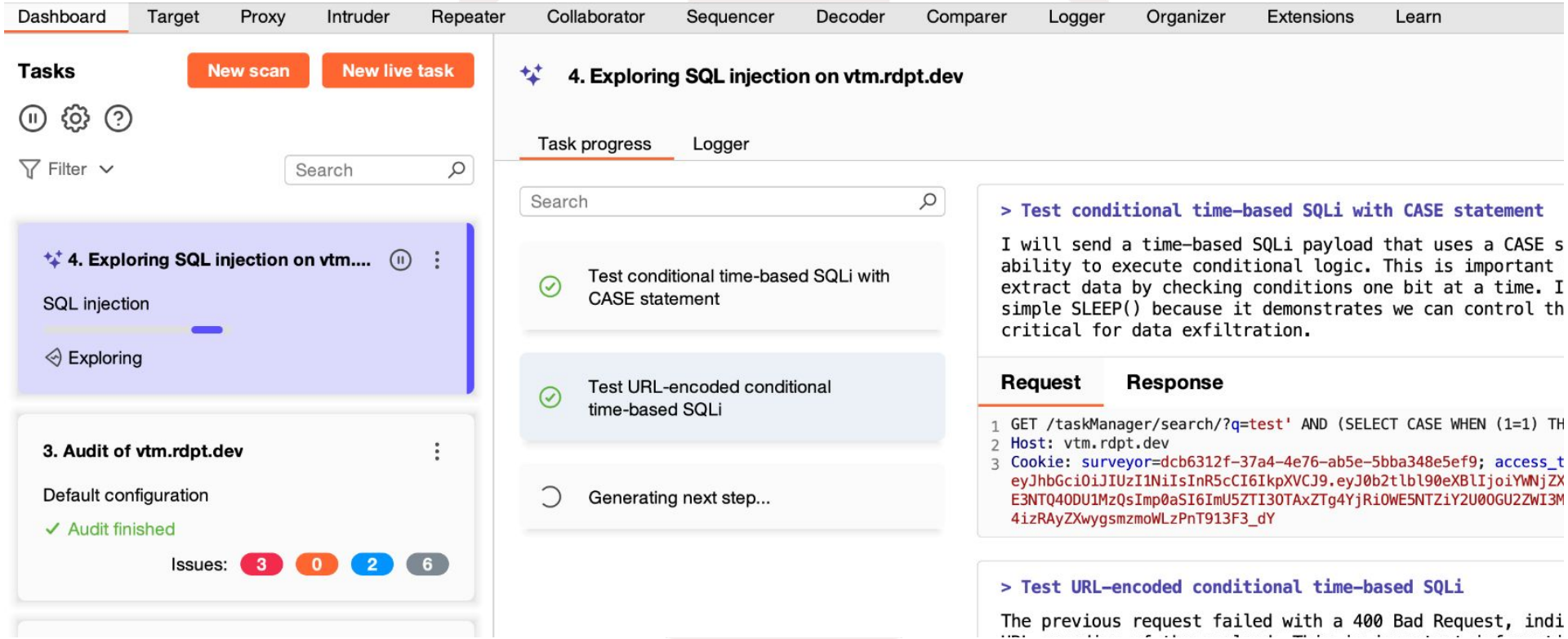
Dynamic AppSec AI - Burp AI

Demo

Dynamic AppSec AI - Burp AI



Dynamic AppSec AI - Burp AI



Exercises: Dynamic AppSec AI

Exercise 1: Identify

Exercise 2: Investigate/Extract

Exercise 3: Prioritize

Exercise: Dynamic AppSec AI (Identify)

🎯Objective: Analyze HTTP session in the provided vector database and identify *all* user-controlled parameters from each request. Aggregate parameters for further analysis

1. Open `scripts/dynamic-identify-parameters.py` and modify the necessary prompt to answer question about parameters.
2. Run the script, note number of endpoints and size of the output
3. Modify the prompt and script to return more results
4. Experiment with prompts and providing contextual clues to improve the results

Exercise: Identify Injection Parameters

Retrospective:

- What locations and parameters were returned?
- What limitations did you note when first running the script?
- What prompt techniques did you utilize to improve the outcome?
- Anything else to share?

Exercise: Dynamic AppSec AI (Investigate/Extract)

🎯Objective: Analyze HTTP session in the provided XML and summarize vulnerabilities for user-controllable parameters

1. Use the provided vtm-session.xml (or Burp XML Export).
2. Open `scripts/dynamic-investigate-parameters.py` and modify the necessary prompt to answer question about parameters.
3. Run the script, note number of endpoints and size of the output
4. Modify the prompt to get better results
5. Change prompts and provide additional context.

Exercise: Investigate/Extract Injection Parameters

Retrospective:

- What were the interesting extracted parameters?
- Were there limitations and false positives?
- What prompt techniques did you utilize to improve the outcome?
- Anything else to share?

Exercise: Dynamic AppSec AI (Prioritize)

🎯**Objective: Analyze HTTP initial injection analysis and prioritize sensitive endpoints for manual analysis.**

1. Open `scripts/dynamic-prioritize-endpoints.py` and modify the necessary prompt to reflect your priorities.
2. Run the script, not output format.
3. Modify the prompt to get expected results or provide additional context (relevant customer data, security concerns, etc).

Exercise: Investigate/Extract Injection Parameters

Retrospective:

- What were the interesting extracted parameters?
- Were there limitations and false positives?
- What prompt techniques did you utilize to improve the outcome?
- Anything else to share?

Source Code Analysis

Source Code Analysis

Lessons Learned:

- Broad questions get fuzzy answers
- Pull Requests/Git Patches: Incorporate both the git patch and full file contents into analysis
- Few Shot Prompting helps guide the LLM
- Mixing deterministic with probabilistic is the best
- Context!
  - Tell the LLM what type of language, library, or web framework (relevant to the type of analysis)
  - When the LLM is blind to a specific tech stack, RAG can get you the rest of the way (so long as the language is supported)

Source Code Analysis

Types of Source Analysis

- Free form: Allow the LLM to reason about what type of vulnerabilities it thinks it is detecting, provide guidance, potential for tuning
- Deterministic: Look for a specific vulnerability, ask discrete code specific questions, use tuning methods

Exercise:
Source Code Review (Free Form)

Exercise: Source Code Review (Free Form)

🎯**Objective: Iterate over all relevant source code files, store results, and then parse thru results to identify highest impact vulnerabilities**

Instructions:

1. Either use vtm or choose an open source project from GitHub
2. Open **`scripts/sca_repo.py`** and modify the necessary bits like the database name and types of file extensions (ours defaults to python)
3. Run the script, allow the results to populate into your FAISS db
4. Now, modify **`scripts/sca_repo_analysis.py`** so that it loads the database of scan results
5. Experiment with prompts and providing contextual clues to improve the results

Exercise: Source Code Review (Free Form)

Retrospective:

- What worked?
- What did not?
- What changes did you make that assisted in improving the results?

Exercise:
Source Code Review (Deterministic-esque)

Exercise: Source Code Review (Deterministic-esque)

🎯**Objective: Iterate over all relevant source code files looking specifically for a specific category of a vulnerability (default is SQL Injection) using Few Shot Prompting to improve accuracy of results**

Instructions:

1. Either use vtm or choose an open source project from GitHub
2. Open **scripts/sca_deterministic_few_shot.py**
3. Look for another category of vulnerability, modify few shot prompt examples, question, and prompt to discover the type of vulnerability you are looking for

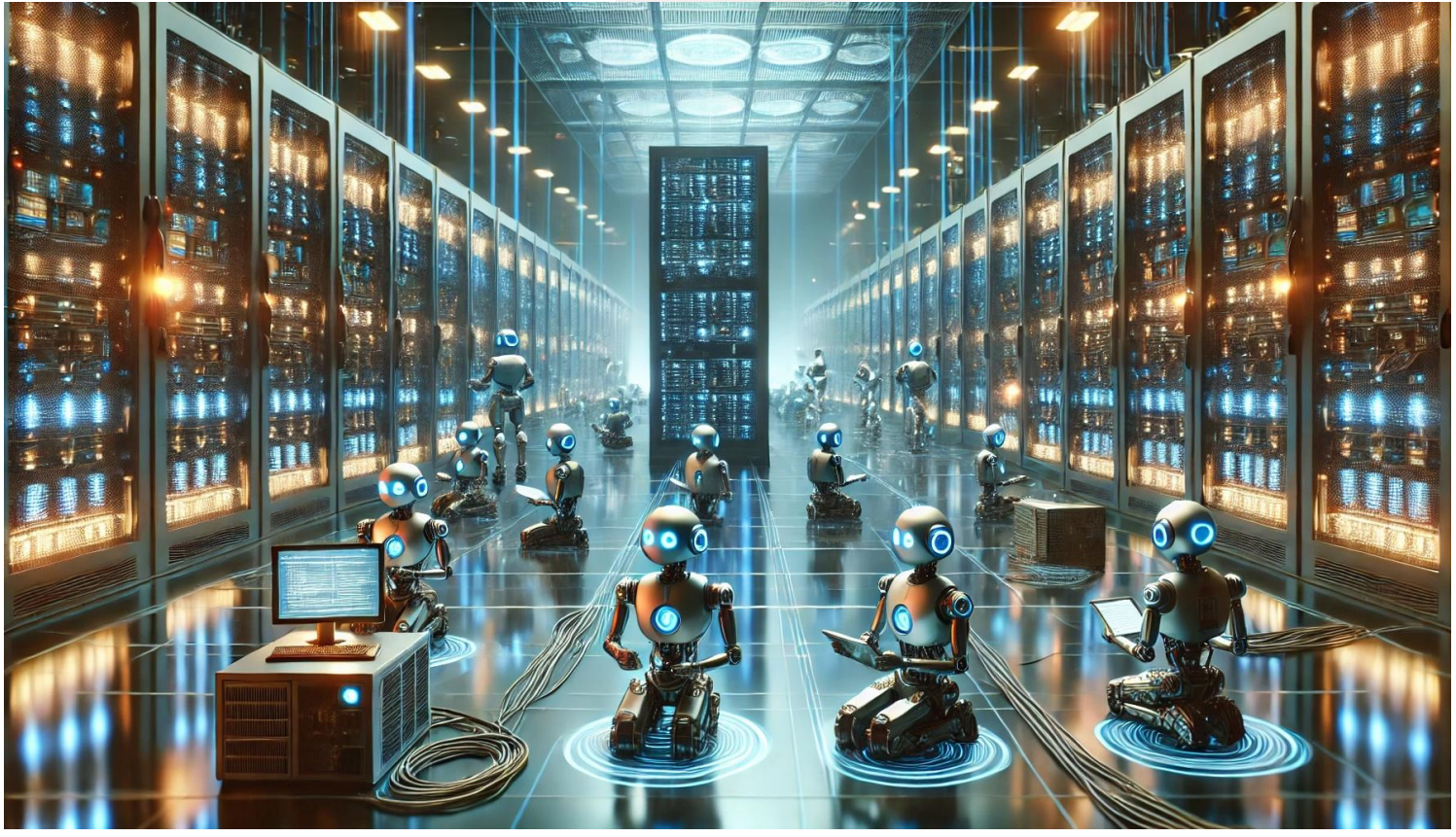
💡Try without few shot prompts first, then, use few shot prompting to improve the results

Exercise: Source Code Review (Deterministic-esqu)

Retrospective:

- What worked?
- What did not?
- What changes did you make that assisted in improving the results?
- Anything else interesting to share?

Agentic AI



Agentic AI

- Refers to providing the LLM with tools it can use as well as a plan to use those tools (ie - “agents”)
- Overcomes:
  - Accuracy issues
  - Context Window Lengths
- You can use the built-in agent options with LangChain
- Platforms exist for production applications to create complex reasoning structures/paths

Agentic AI

Langchain built-in Agents (VERSION 1.0!)

https://python.langchain.com/v0.1/docs/modules/agents/agent_types/

Agent Type	Intended Model Type	Supports Chat History	Supports Multi-Input Tools	Supports Parallel Function Calling	Required Model Params	When to Use	API
Tool Calling	Chat	✅	✅	✅	`tools`	If you are using a tool-calling model	Ref
OpenAI Tools	Chat	✅	✅	✅	`tools`	[Legacy] If you are using a recent OpenAI model (`1106` onwards). Generic Tool Calling agent recommended instead.	Ref
OpenAI Functions	Chat	✅	✅		`functions`	[Legacy] If you are using an OpenAI model, or an open-source model that has been finetuned for function calling and exposes the same `functions` parameters as OpenAI. Generic Tool Calling agent recommended instead	Ref

Agentic AI

Langchain built-in Agents (VERSION 1.0!)
https://python.langchain.com/v0.1/docs/modules/agents/agent_types/

XML	LLM	✅				If you are using Anthropic models, or other models good at XML	Ref
Structured Chat	Chat	✅	✅			If you need to support tools with multiple inputs	Ref
JSON Chat	Chat	✅				If you are using a model good at JSON	Ref
ReAct	LLM	✅				If you are using a simple model	Ref
Self Ask With Search	LLM					If you are using a simple model and only have one search tool	Ref

Agentic AI

(Reason + Act) = ReAct

Explanation https://react-lm.github.io/

Research material
https://research.google/blog/react-synergizing-reasoning-and-acting-in-language-models/

ReAct (Reason + Act):
ReAct is a framework for task-solving in LLMs that combines reasoning and action. It encourages the model to first "reason" by explaining its thought process, then "act" by performing an action or generating a response. This iterative process helps break down complex problems, verify assumptions, and refine outputs for accurate and logical results.

Agentic AI

```python
class SearchInput(BaseModel):
    query: str = Field(description="should be a search query")

class CustomSearchTool(BaseTool):
    name: str = "custom_search"  # Explicitly type all attributes
    description: str = "Useful for when you need to answer questions about code"
    args_schema: Type[SearchInput] = SearchInput  # Explicitly set schema


    def _run(
        self, query: str, run_manager: Optional[CallbackManagerForToolRun] = None
    ) -> str:
        """Use the tool."""
        # Repo details to clone
        faiss_db_path = "../vector_databases/vtm_faiss"
        db = FAISS.load_local(
            faiss_db_path,
            BedrockEmbeddings(model_id='amazon.titan-embed-text-v1'),
            allow_dangerous_deserialization=True
        )
        val =  db.similarity_search(query)
        return val
        #return "LangChain"

    async def _arun(
        self, query: str, run_manager: Optional[AsyncCallbackManagerForToolRun] = None
    ) -> str:
        """Use the tool asynchronously."""
        raise NotImplementedError("custom_search does not support async")
```

```python
tools = [CustomSearchTool()]
```

Agentic AI

```
You have access to the following tools:

{tools}

To use a tool, please use the following format:

```
Thought: Do I need to use a tool? Yes
Action: the action to take, should be one of [{tool_names}]
Action Input: the input to the action
Observation: the result of the action
```

When you have a response to say to the Human, or if you do not

```
Thought: Do I need to use a tool? No
Final Answer: [your response here]
```

Begin!

Previous conversation history:
{chat_history}

New input: {input}
{agent_scratchpad}
```

```
agent_executor.invoke({"input": input, "chat_history": ""})
```

The `{agent_scratchpad}` placeholder in the prompt is a dynamic field that gets replaced with the intermediate reasoning or action steps that the agent generates during its thought process.

Exercise:
Agentic AI (Basic)

Exercise: Agentic AI (Basic)

🎯Objective:

Instructions:

1. Run `scripts/agentic_basic.py`
2. Modify the script to:
    a. Retrieve background information about the application
    b. Provide that information as context
    c. Modify the prompt as you desire to find more issues
    d. Tune this approach using Few Shot Prompt Examples

Deepseek Model ID for AWS Bedrock

us.deepseek.r1-v1:0

us.anthropic.claude-3-5-sonnet-20241022-v2:0

Exercise: Agentic AI (Basic)

Retrospective:

- What worked?
- What did not?
- What changes did you make that assisted in improving the results?
- Anything else interesting to share?

Agentic Advanced

LangGraph + ReAct = 💙

💙 Why it works

- Flexible reasoning + structured control
- Observable, testable workflows
- Resilient to LLM flakiness

⚠️ Watch for

- Prompt leakage
- Agent loops / runaway cost
- Over-privileged tools

Agentic AI

LangGraph Benefits ->

Here are some common issues that arise in complex deployments, which LangGraph Platform addresses:

- **Streaming Support**: As agents grow more sophisticated, they often benefit from streaming both token outputs and intermediate states back to the user. Without this, users are left waiting for potentially long operations with no feedback. LangGraph Server provides multiple streaming modes optimized for various application needs.
- **Background Runs**: For agents that take longer to process (e.g., hours), maintaining an open connection can be impractical. The LangGraph Server supports launching agent runs in the background and provides both polling endpoints and webhooks to monitor run status effectively.
- **Support for long runs**: Vanilla server setups often encounter timeouts or disruptions when handling requests that take a long time to complete. LangGraph Server's API provides robust support for these tasks by sending regular heartbeat signals, preventing unexpected connection closures during prolonged processes.
- **Handling Burstiness**: Certain applications, especially those with real-time user interaction, may experience "bursty" request loads where numerous requests hit the server simultaneously. LangGraph Server includes a task queue, ensuring requests are handled consistently without loss, even under heavy loads.
- **Double Texting**: In user-driven applications, it's common for users to send multiple messages rapidly. This "double texting" can disrupt agent flows if not handled properly. LangGraph Server offers built-in strategies to address and manage such interactions.
- **Checkpointers and Memory Management**: For agents needing persistence (e.g., conversation memory), deploying a robust storage solution can be complex. LangGraph Platform includes optimized checkpointers and a memory store, managing state across sessions without the need for custom solutions.
- **Human-in-the-loop Support**: In many applications, users require a way to intervene in agent processes. LangGraph Server provides specialized endpoints for human-in-the-loop scenarios, simplifying the integration of manual oversight into agent workflows.

```python
class AgentState(TypedDict):
    input: str
    output: str

def agent_node(state: AgentState) -> AgentState:
    """Run the ReAct agent"""
    result = agent_executor.invoke({"input": state["input"]})
    return {"input": state["input"], "output": result["output"]}

# Create simple LangGraph workflow
workflow = StateGraph(AgentState)
workflow.add_node("agent", agent_node)
workflow.set_entry_point("agent")
workflow.add_edge("agent", END)

# Compile the graph
langgraph_app = workflow.compile()

def analyze_code_with_langgraph(input_code: str) -> dict:
    """Analyze code using LangGraph wrapper around ReAct agent"""
    result = langgraph_app.invoke({"input": input_code})
    return result      You, 13 minutes ago • agentic langgraph and
```

STEP THRU CODE

Exercise:
Agentic AI (Advanced)

Exercise: Agentic AI (Advanced)

🎯Objective:

Instructions:

1. Run `scripts/sast/langgraph_react_demo.py`
2. Modify the script to:
    a. Improve security findings
    b. Improve the reporting format
    c. Optionally, add a new step to use output of one prompt into another for report generation
    d. Optionally, build a new tool to use and add it to the tools list

Exercise: Agentic AI (Advanced)

Retrospective:

- What worked?
- What did not?
- What changes did you make that assisted in improving the results?
- Anything else interesting to share?

Logging and Monitoring

Logging & Monitoring AI - Traffic Analysis

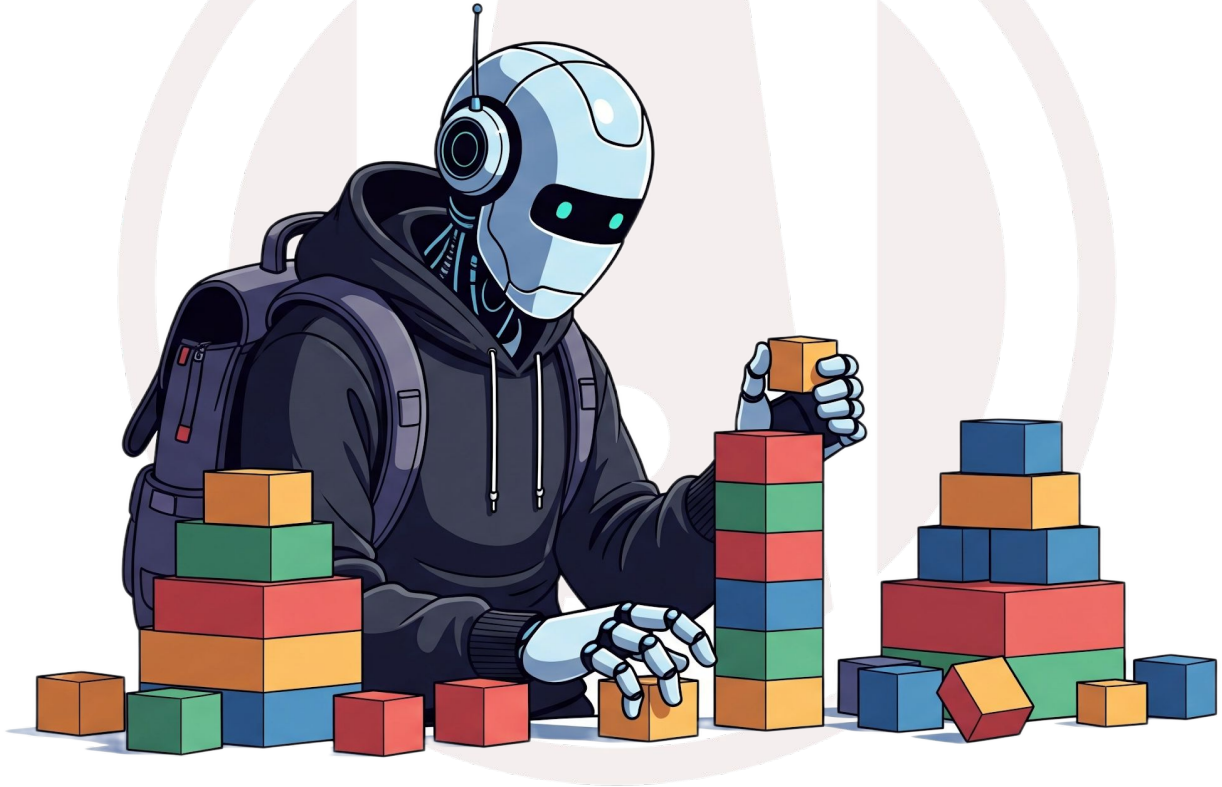
- Application Logs and Interactions
- Server Logs
- Integration with SIEM tools

Exercise: L&M - AppSec Attack Monitoring

🎯**Objective: Examine the vtm-monitor-session.xml for possible injection attacks.**

1. `scripts/monitor-injection-attacks.py`
2. View prompts for possible weaknesses and confusion.
3. Modify the prompt to improve results, include few-shots prompts and reflection.
4. Run updated script.
5. Continue to iterate to detect additional attacks and eliminate false positives

Software Composition Analysis



SCA AI - Utilizing an AI IDE

Summarization and Identification are key

Utilize AI IDE plugins to quickly get the lay of the land.

Custom Prompts to go further.

Consider RAG or MCP of Known Vulnerable Components

Exercise: AI IDE Analysis

Activity (with Windsurf, run against bridgetroll or vtm):

- Identify security-impacting libraries from project dependencies.
- Identify known CVEs or possible weaknesses with all libraries.
- Trace the exploitability of the above weaknesses.

Retrospective:

- What worked?
- What did not?
- What changes did you make that assisted in improving the results?
- Anything else interesting to share?

Testing LLMs

Testing LLMs - Background

- LLMs used to build out and run integration and unit tests for tuning
- Create repeatable results if and when you change variables that impact the LLM results.
- Even small changes in context, prompts, etc. can have big impacts
- Important if you want to change/swap/upgrade LLMs

Testing Framework

1. Use PyTest
2. Other options exist including:
    a. https://python.langchain.com/docs/concepts/testing/
    b. https://github.com/openai/evals
    c. https://github.com/confident-ai/deepeval

Testing Framework

4 important components:

1. Makefile (specify what “make test” does)
2. pytest.ini (specify test markers)
3. Test file (**ex: agentic_test.py**) ends in “_test.py”
4. Use the marker **@pytest.mark.integration** for integration tests

```
@pytest.mark.integration
def test_analyze_code_insecure():
```

***Note that we use unit-test to test structures (json, yaml), and integration for behaviors**

Exercise:
Testing

Exercise: Testing

🎯Objective:

Instructions:

1. Open `scripts/agentic_test.py`
2. Add an integration test (hint: you can make one that checks for insecure == false)
3. Bonus points - Create a unit-test that checks JSON or other structure for validity

Exercise: Agentic AI (Basic)

Retrospective:

- Any issues?
- Did anyone create a unit-test?
- Anything else to share/ask?

Threat Modeling

Threat Modeling

Threat Modeling with LLMs

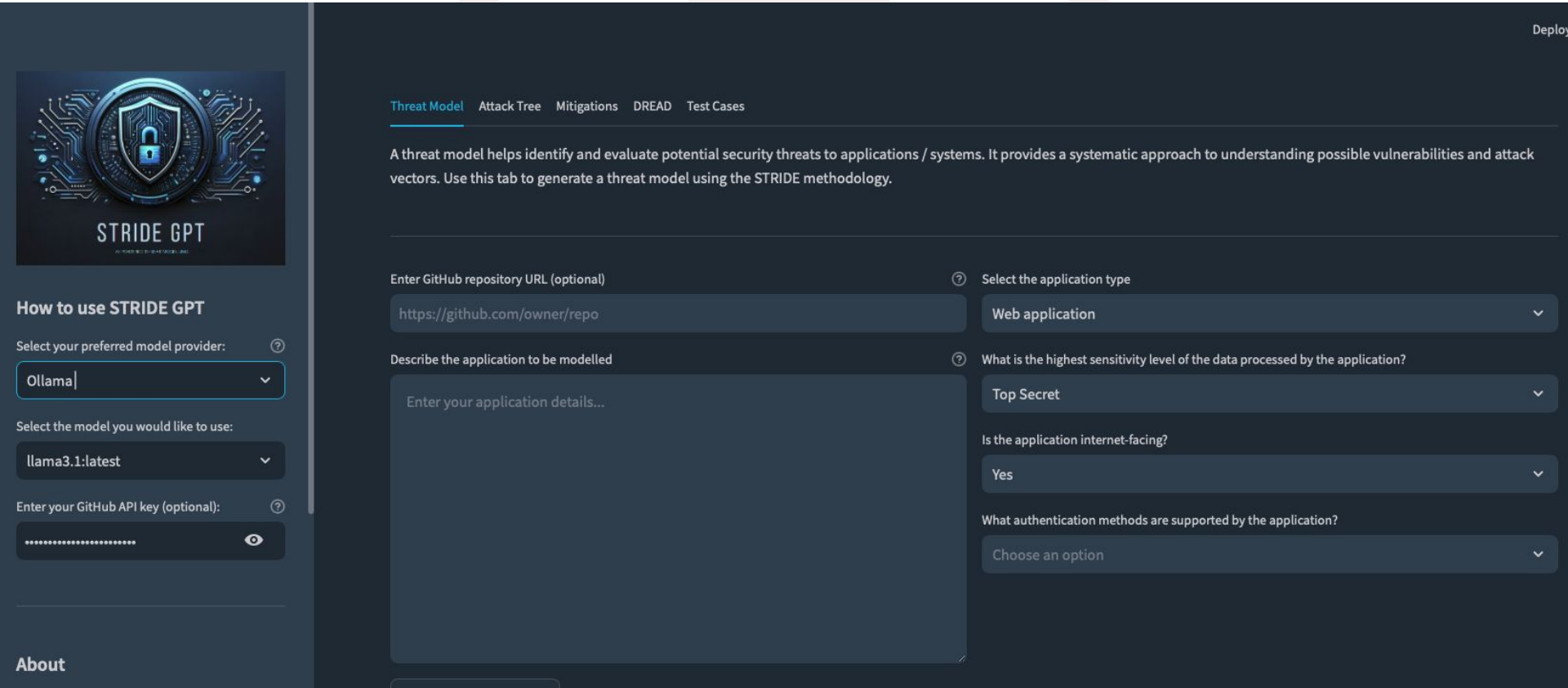
- Creativity built in to LLMs is useful for Threat Modeling
- Initially just used Chat GPT to ask questions about threat, utilizing knowledge gleaned during initial Source Code Review process.
- Fell back on STRIDE and DREAD for classification
- Then it was codified.

Threat Modeling - Stride GPT

“STRIDE GPT is an AI-powered threat modelling tool that leverages Large Language Models (LLMs) to generate threat models and attack trees for a given application based on the STRIDE methodology.”

https://github.com/mrwadams/stride-gpt

Threat Modeling - Stride GPT



Threat Modeling - Stride GPT - BHIMA

Enter GitHub repository URL (optional)

https://github.com/owner/repo

Describe the application to be modelled

BHIMA
==========

BHIMA is a free, open source accounting and hospital information management system
(HIMS) tailored for rural hospitals in the Democratic Republic of the Congo. We are an international team based all over the world.

BHIMA is an acronym for _basic hospital information management application_. It was originally
developed by [IMA World Health](https://imaworldhealth.org/) with funding from the Foreign Commonwealth and Development Office (FCDO).

Generate Threat Model

Select the application type

Web application

What is the highest sensitivity level of the data processed by the application?

Confidential

Is the application internet-facing?

Yes

What authentication methods are supported by the application?

Basic ×

Threat Modeling - Stride GPT - BHIMA Threats

Threat Model

Threat Type	Scenario	Potential Impact
Spoofing	Attacker creates a fake login page and steals users' credentials by manipulating the Basic authentication method used in BHIMA.	Unauthorized access to sensitive data, including patient and financial information.
Spoofing	An attacker sends a phishing email to hospital administrators with a link to a fake version of the BHIMA login page, tricking them into entering their credentials.	Compromise of sensitive data and potential malware infection.
Tampering	An attacker intercepts and modifies patient billing information while it's being transmitted between the client and server, causing incorrect charges to be applied.	Financial loss for patients and hospitals due to tampered data.
Tampering	A malicious user hacks into the Redis session management system and alters the login credentials of other users, allowing them to access sensitive areas of the application.	Unauthorized access to sensitive areas of the application and potential data tampering.
Repudiation	A user makes changes to their account settings without actually performing the actions, then denies responsibility for the alterations when confronted about it.	Loss of trust in the system and potentially, legal issues due to lack of accountability.
Repudiation	An attacker uses BHIMA's reporting plugins to fabricate fake reports, then denies responsibility for the malicious activity.	Loss of trust in the system and potentially, legal issues due to lack of accountability.
Information Disclosure	BHIMA is hacked, and sensitive data such as patient records or financial information is leaked online.	Compromise of sensitive data and potential reputational damage for the hospital and BHIMA.

Threat Modeling - Stride GPT - BHIMA Threats

Information Disclosure	BHIMA is hacked, and sensitive data such as patient records or financial information is leaked online.	Compromise of sensitive data and potential reputational damage for the hospital and BHIMA.
Information Disclosure	An attacker exploits a vulnerability in BHIMA's code to gain access to server logs, revealing sensitive information about user activity.	Compromise of sensitive data and potential reputational damage for the hospital and BHIMA.
Denial of Service	A large number of requests flood the server, causing it to become unresponsive and preventing legitimate users from accessing their accounts.	Loss of productivity due to system downtime and potential financial loss for hospitals.
Denial of Service	An attacker uses a botnet to send numerous requests to BHIMA's server, overwhelming it with traffic and causing it to crash.	System downtime, financial loss due to lost productivity, and potential reputational damage.
Elevation of Privilege	A user gains elevated privileges within BHIMA by exploiting a vulnerability in the NodeJS server-side code, allowing them to access sensitive areas of the application.	Unauthorized access to sensitive areas of the application and potential data tampering.
Elevation of Privilege	An attacker discovers a backdoor in BHIMA's Redis session management system, allowing them to gain elevated privileges within the application.	Unauthorized access to sensitive areas of the application and potential data tampering.

Improvement Suggestions

- Implement additional security measures for user authentication, such as multi-factor authentication or more secure password storage.
- Regularly update dependencies in the NodeJS code to prevent vulnerabilities from being exploited by attackers.
- Use a Web Application Firewall (WAF) to monitor and filter incoming traffic to the application, preventing common web attacks.

Threat Modeling - Stride GPT - BHIMA Attack Trees

Generate Attack Tree

Attack Tree Code:

```
graph TD
    A["Compromise Web Application (BHIMA)"] --> B("Spoofing")
    B --> C["Impersonate User Credentials"]
    B --> D["Bypass Basic Authentication"]

    A --> E("Tampering")
    E --> F["Modify Sensitive Data in Transit"]
    E --> G["Alter Data in MySQL Database"]

    A --> H("Repudiation")
    H --> I["Log Tampering in NodeJS Server"]
    H --> J["Modify Redis Session Data"]

    A --> K("Information Disclosure")
    K --> L("Data Leak via HTTP")
    K --> M("Sensitive Data Exposure in MySQL")

    A --> N("Denial of Service (DoS)")
    N --> O["Overload Redis with Sessions"]
```

Threat Modeling - Stride GPT - Mitigations

Threat Mitigation Strategies

Threat Type	Scenario	Suggested Mitigation(s)
Spoofing	Attacker creates a fake login page and steals users' credentials by manipulating the Basic authentication method used in BHIMA.	Implement a robust authentication mechanism, such as OAuth or OpenID Connect, to reduce reliance on Basic authentication. Utilize Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates for encryption and identity verification.
Spoofing	An attacker sends a phishing email to hospital administrators with a link to a fake version of the BHIMA login page, tricking them into entering their credentials.	Educate users on phishing techniques and provide training on identifying suspicious emails and links. Implement email authentication mechanisms like SPF, DKIM, or DMARC to prevent spoofed emails.
Tampering	An attacker intercepts and modifies patient billing information while it's being transmitted between the client and server, causing incorrect charges to be applied.	Use Transport Layer Security (TLS) encryption for data in transit and verify certificates of all parties involved. Implement a digital signature scheme, such as HMAC or ECDSA, to ensure data integrity during transmission.
Tampering	A malicious user hacks into the Redis session management system and alters the login credentials of other users, allowing them to access sensitive areas of the application.	Regularly update Redis sessions with secure random values, ensuring each user has unique credentials. Implement Redis security features like password authentication, ACLs (Access Control Lists), or REDIS_CLUSTER to prevent unauthorized access.

Threat Modeling - Stride GPT - DREAD Risk Assessment

Threat Type	Scenario	Damage Potential	Reproducibility	Exploitability	Affected Users	Discoverability	Risk Score
Spoofing	Attacker creates a fake login page and steals users' credentials by manipulating the Basic authentication method used in BHIMA.	9	8	7	10	6	8.00
Spoofing	An attacker sends a phishing email to hospital administrators with a link to a fake version of the BHIMA login page, tricking them into entering their credentials.	8	7	6	9	5	7.00
Tampering	An attacker intercepts and modifies patient billing information while it's being transmitted between the client and server, causing incorrect charges to be applied.	8	6	5	9	7	7.00
Tampering	A malicious user hacks into the Redis session management system and alters the login credentials of other users, allowing them to access sensitive areas of the application.	9	8	7	10	6	8.00
Repudiation	A user makes changes to their account settings without actually performing the actions, then denies responsibility for the alterations when confronted about it.	7	5	4	8	6	6.00
Repudiation	An attacker uses BHIMA's reporting plugins to fabricate fake reports, then denies responsibility for the malicious activity.	8	7	6	9	5	7.00

Threat Modeling - Stride GPT - Test Cases

Spoofing - An attacker sends a phishing email to hospital administrators with a link to a fake version of the BHIMA login page, tricking them into entering their credentials

```
**Test Case: Phishing Email**
Given an attacker has sent a phishing email with a link to a fake BHIMA login page to a hospital administrator
And the administrator clicks on the link and enters their credentials
Then the attacker should have obtained the administrator's credentials
```

Tampering - An attacker intercepts and modifies patient billing information while it's being transmitted between the client and server, causing incorrect charges to be applied

```
**Test Case: Modified Patient Billing Information**
Given an attacker has intercepted patient billing information being transmitted between the client and server
And the attacker modifies the billing information to reflect incorrect charges
Then the system should apply the incorrect charges to the patient's account
```

Tampering - A malicious user hacks into the Redis session management system and alters the login credentials of other users, allowing them to access sensitive areas of the application

Exercise:
Threat Model

Exercise: Threat Model

🎯**Objective: Create a Threat Model of an Open Source Project**

Instructions:

1. Clone and follow instructions for running Stride GPT (https://github.com/mrwadams/stride-gpt)
2. Choose an open source project from Github (e.g. https://github.com/railsbridge/bridge_troll)
3. Copy and paste the raw README file from the chosen project into Stride GPT.
4. ***** Note, if you navigate away from a tab it clears out any generated content. Copy/Paste if you want to save it.**
5. Utilize Ollama + llama3.1 to generate Threats, Attack Trees (sometimes), Mitigations, Risk Assessment and Test Cases
6. Utilize Ollama + mistral to generate Threats, Attack Trees (sometimes), Mitigations, Risk Assessment and Test Cases
7. Review output for appropriateness, analysis, note good and bad

Exercise: Threat Model

Retrospective:

- What worked?
- What did not?
- Which model worked better for Threats, Attack Trees, Mitigations, DREAD, and Test Cases?