Hands On with AI: Using LLMs to Detect IDOR and Auth Flaws

CONFIDENTIAL

DRYRUN .SECURITY

Introductions

DryRun Security founded in 2022

James Wickett CEO, Ken Johnson CTO

"DryRun Security finds the needle in the haystack of code changes, so AppSec teams spot unknown risks before they start."

My Background (Ken):

- Application Security @ GitHub, ~6 years
- Co-Host of Absolute AppSec
- Tribe of Hackers
- Train on Secure Code Review & Harnessing LLMs for AppSec
- Been a practitioner of AppSec since 2008

Overview

Introductions

IDOR Background & Problem Statement

Agentic AI

LangChain

Components

Code Walk-Thru / Demo

Use Cases

Final Thoughts

IDOR

Insecure Direct Object Reference (IDOR) is a common web application vulnerability where an application directly references internal objects—such as database records or files—without proper access controls. As a result, attackers can manipulate these references (like changing a URL parameter) to access or modify data they shouldn't have permission to view or alter.

IDOR

Example (from Django application):

```
def update_user_active(request):
    user_id = request.GET.get('user_id')
    User.objects.filter(id=user_id).update(is_active=False)
```

IDOR

With Authorization Decorators:

```
@login_required
@user_passes_test(can_create_project)
def update_user_active(request):
    user_id = request.GET.get('user_id')
    User.objects.filter(id=user_id).update(is_active=False)
```

IDOR

Dilemna:

The LLM does not know what `can_create_project` does

It *might* know what **login_required** or **user_passes_test** does, depending on the training data

Solutions

Tell the LLM exactly what that function does

- Provided as context
- Provided as an example
- **Give the LLM the ability to search the code to review the function**

Agentic AI

Agentic AI

Agentic AI refers to autonomous intelligent systems that can initiate actions, pursue goals, and adapt to changing conditions without requiring continuous human intervention. They have a sense of "agency," allowing them to reason, make decisions, and carry out tasks on their own.

Agentic AI

crewai

Two routes to using Agentic AI:

- Use an agentic AI service
- **Build your own**

Reason + Act = ReAct

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.

Reason + Act = ReAct

Suggested Reading material:

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

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

https://medium.com/@aydinKerem/which-ai-agent-framework-i-should-use-crewai-langgraph-majestic-one-and-pure-code-e16a6e4d9252

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

Important components for today:

- **Embedding & Vector storage:** Used to make source code searchable
- **Prompting:** Directions for the LLM
- **Context:** Helpful background information/data
- **ReAct / AgentExecutor:** Orchestration behind pseudo-reasoning

Components

Embeddings & Vector Storage

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.

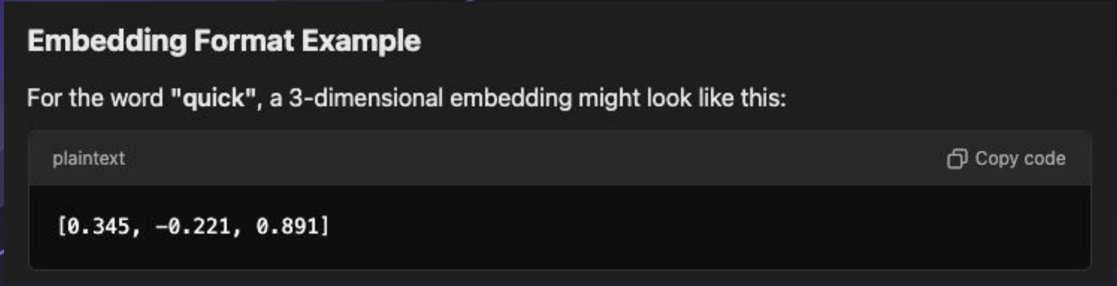
- Dense Representation - Fixed Size
- Semantic Encoding - Relationships between things
- Domain Specific - Can be pre-trained

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!

Embeddings & Vector Storage

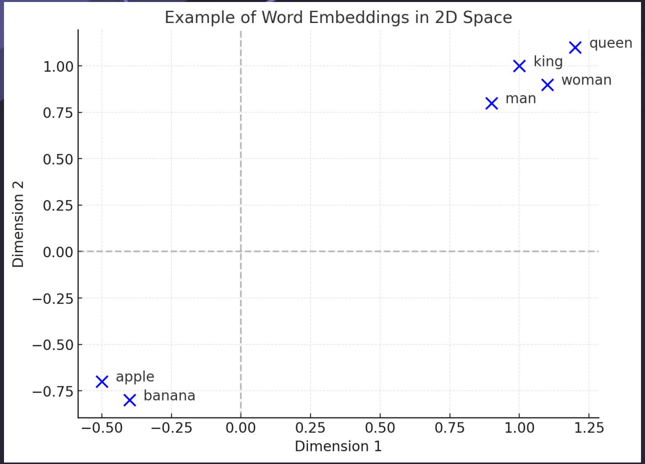


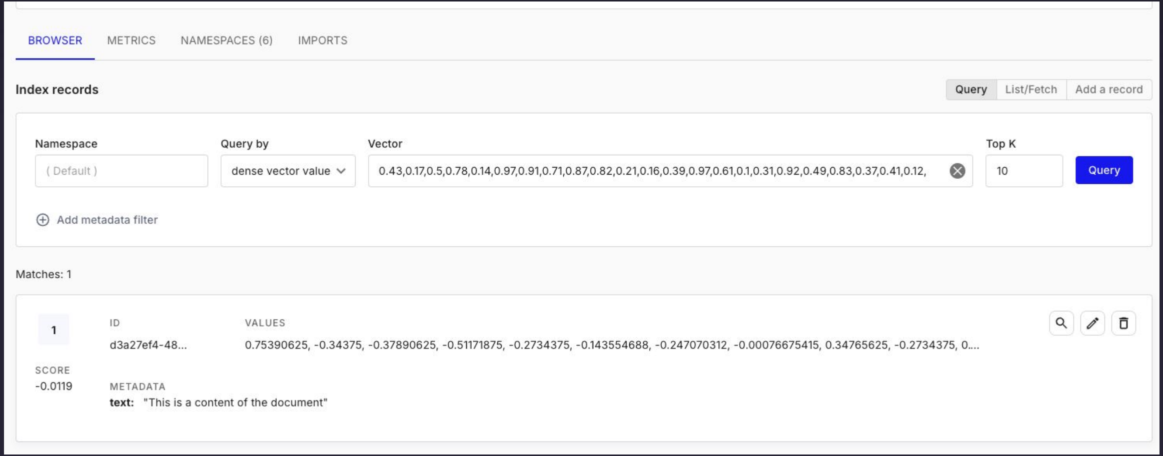
Embedding Format Example

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

plaintext

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





Index records

Namespace: (Default)
Query by: dense vector value
Vector: 0.43,0.17,0.5,0.78,0.14,0.97,0.91,0.71,0.87,0.82,0.21,0.16,0.39,0.97,0.61,0.1,0.31,0.92,0.49,0.83,0.37,0.41,0.12,
Top K: 10

Matches: 1

	ID	VALUES
1	d3a27ef4-48...	0.75390625, -0.34375, -0.37890625, -0.51171875, -0.2734375, -0.143554688, -0.247070312, -0.00076675415, 0.34765625, -0.2734375, 0....

SCORE
-0.0119

METADATA
text: "This is a content of the document"

Prompting

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

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

Prompting

ReAct Prompt Engineering Principles

- **Thought:** What task do I need to accomplish?
- **Action:** What can I use to help accomplish this goal?
- **Observation:** What was the outcome of the action I took and what is my analysis?

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

Context comes in many forms:

- 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
- Large portions of source code
- Call Graphs / Abstract Syntax Tree (AST)
- Directory listings
- Framework or language information
- System or application logs
- Output of Scanner Findings

ReAct / AgentExecutor

- AgentExecutor is a utility in LangChain that manages the sequence of ReAct steps for an agent
- It orchestrates the "Thought → Action → Observation → Thought → ..." loop until the agent reaches a final answer.
- Supports building custom tools
- Custom tools means that you can build tooling to do whatever it is you need the LLM to do
- 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.

Code Walk-Thru / Demo

Use Cases

Use Case #1

GraphQL Authorization Nuanced Authz Flaw

- Can fail in one of several ways:
    - Authorization related function is missing
    - Authorization functionality improperly configured
    - Authorization functionality improperly built but other authorization in place
    - If none of the above, does it inherit from a known safe class

Use Case #2

Nuanced REST API Authz flaw

- Minor differences in the way that the authorization function is invoked can make it either effective or fail completely
- Two levels of abstractions when invoking, can be either, has to be invoked in specific ways
- Interested in reviewing any changes to endpoints but only interested in blocking new endpoint definitions that contain the authz flaw

Use Case #3

IDOR in Service Level Objects

- Non traditional patterns of IDOR (doesn’t match the generic framework examples that default SAST rules would recognize)
- Various nuanced ways that actor permissions are not checked against resource access - **not something that can be searched for using patterns/grep/regex**

Final Thoughts

Final Thoughts

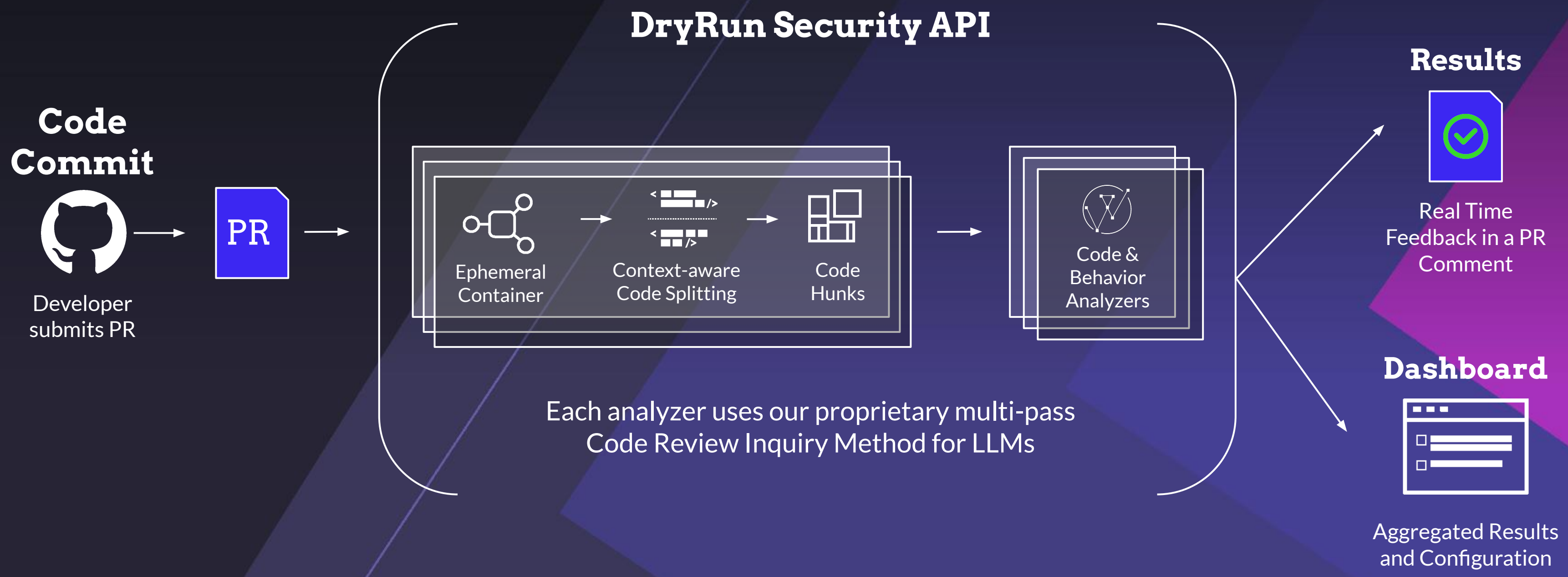
- AI is a powerful tool for complex analysis
- Agentic AI empowers the LLM to get the information it needs to perform even more complex analysis
- Authorization flaws remain chief amongst security practitioners concerns
- To date SAST tooling has not had the capability to perform the level of complex analysis required for many reasons
- Embrace the future!

Final Thoughts

- DryRun Security makes AI easy for AppSec Pros
- Custom Policies are a way for us to accomplish finding nuanced vulnerabilities
- Whether you want to build your own solution or use DryRun Security, AI will transform your detection & prevention capabilities

Our architecture at a glance

DRYRUN .SECURITY


Code Commit
Developer submits PR
PR
DryRun Security API
Ephemeral Container
Context-aware Code Splitting
Code Hunks
Code & Behavior Analyzers
Results
Real Time Feedback in a PR Comment
Dashboard
Aggregated Results and Configuration

Each analyzer uses our proprietary multi-pass Code Review Inquiry Method for LLMs

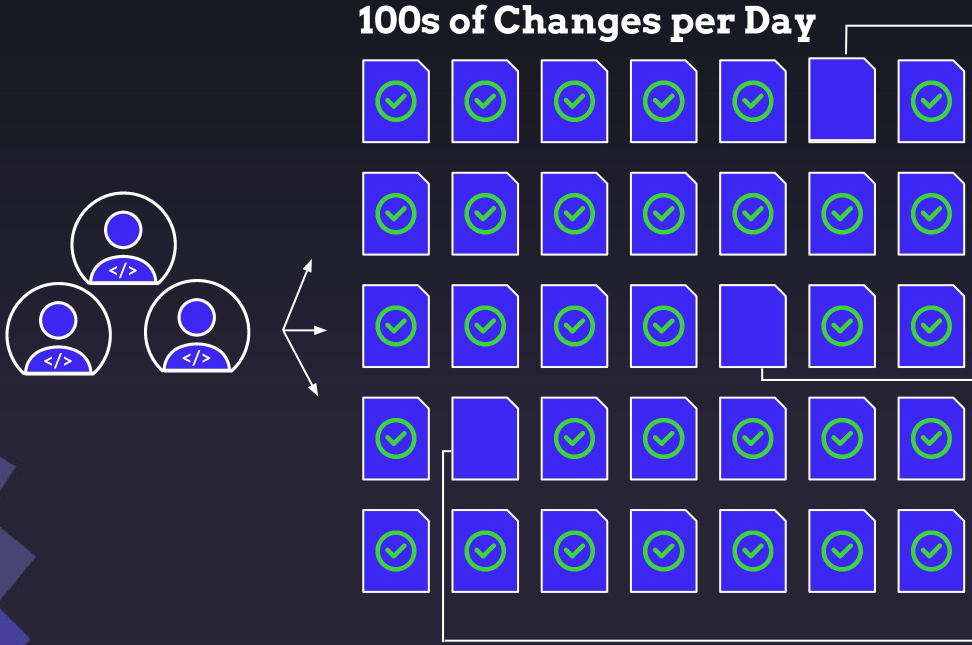
Analyzer Accuracy Assured

- Code Context
- Scoping & Isolation
- DryRun Security Knowledgebase
- Positive and Negative Tuning
- Private LLM

[Code Review Inquiry Method]

DRYRUN .SECURITY

Which code changes are the riskiest?



Modifies encryption for sensitive data

Updates the password reset flow on the endpoint `/admin-login`

Grants new permissions to `admin` on API endpoint `/v1/list/users`

Advance your AppSec Practice

- ✓ Know which code changes are the most risky
- ✓ Get your security team out of the rule maintenance game
- ✓ Every repo protected, every PR analyzed
- ✓ Scale security without adding headcount

WORKSHOP

Effective AI Prompting Tricks for the Busy CISO

with James Wickett

Register