Teaching AI to Fix Container Vulnerabilities

Anton Sankov

Anton Sankov

Staff AI Engineer at Cast AI

- Doing Container Security stuff since 2020
- Currently working on AI agents

Anton Sankov
asankov.dev

Agenda

1. What is an AI Agent
2. Attempt #1
3. Lessons Learned and Attempt #2
4. Will the AI Replace Us

DISCLAIMER:

This talk covers a commercial non-open source product.

This is not a sales pitch, it’s just a knowledge and experience sharing session.


AI AGENTS
SO HOT RIGHT NOW
imgflip.com

What is an AI Agent?

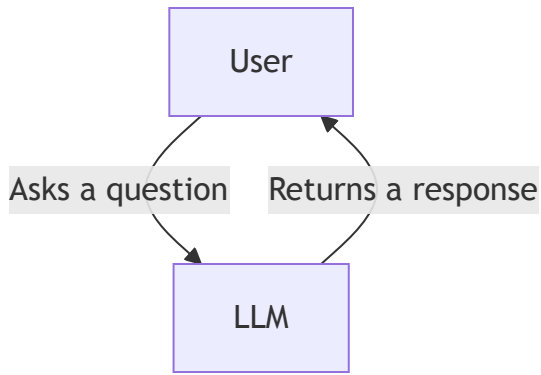
AI agents are software systems that use AI to pursue goals and complete tasks on behalf of users.

They show reasoning, planning, and memory and have a level of autonomy to make decisions, learn, and adapt.

https://cloud.google.com/discover/what-are-ai-agents

Chat vs Agent

Chat



Execution time: 1 - 30 seconds

Chat vs Agent

Agent

Execution time: 5 - 60 minutes

The Problem

Endless lists of vulnerabilities

The Solution

Build an AI agent that remediates security vulnerabilities (CVEs) in container images.

Input:

- container image
- git repository (where the source of the container image is)
- a list of vulnerabilities inside the container image
  - system (Linux packages, binaries, etc.)
  - application (Java/JS/Python/Go, etc. libraries and dependencies)

Expected output:

- a Pull Request with changes that remediate vulnerabilities in that image

Attempt #1

Use multiple AI agents, but orchestrate everything in code

```
for image in vulnerable_images:
    # an AI agent that finds the Dockerfile in the code repo
    dockerfile = find_dockerfile(image.git_repo)

    # an AI agent that remediates vulnerabilities in system(OS) packages
    remediate_system_vulnerabilities(image.vulnerabilities, dockerfile)

    # an AI agent that finds the application dependencies file (package.json, pom.xml, build.gradle, etc.)
    dependency_file = find_dep_file(image.git_repo)

    # an AI agent that remediates vulnerabilities in the application dependencies
    remediate_application_vulnerabilities(image.vulnerabilities, dependency_file)

    # an AI agent that finds the build command and builds the image with the new changes
    new_image = build_image(image.git_repo)

    # [deterministic] scan the newly build image
    scan_results = scan_image(new_image)

    # MORE DETERMINISTIC LOGIC - compare results, commit changes, open a PR, etc. ...
```

Attempt #1

Fallacies of that approach

Each one of these agents we invoke is a potential point of failure. Upon failure of one agent, we failed the whole thing. Probability of failure is:

$$1 - (1 - M)^N$$

where:

M	probability of failure of 1 agent
N	number of agents

Attempt #1

Fallacies of that approach

Sometimes the build would fail, because of lack of resources

(looking at you, Java apps)

Attempt #1

Fallacies of that approach

The agent would produce changes for images that the users don't care about

Attempt #1

Fallacies of that approach

Sometimes the agent would just produce nonsense

Attempt #1

Nonsense like this:

```
+ # ---- Patch CVE: upgrade bundled pip (24.0 → 24.1.2) ----
+ # Note: The bundled Python in Google Cloud SDK is glibc-linked and may not run on Alpine (musl).
+ # To ensure the bundled pip is upgraded without executing the bundled interpreter, we fetch
+ # the pip 24.1.2 wheel and extract it directly into the bundled site-packages directory.
+ RUN set -eux; \
+     PIP_VERSION=24.1.2; \
+     SITE_PKGS=/google-cloud-sdk/platform/bundledpythonunix/lib/python3.11/site-packages; \
+     mkdir -p "${SITE_PKGS}"; \
+     PIP_WHL_URL=$(curl -s "https://pypi.org/pypi/pip/${PIP_VERSION}/json" | jq -r '.urls[] | select(.filename=="pip-'"${PIP_VERSION}"'-py3-none-any.whl") | .url'); \
+     wget -O /tmp/pip-${PIP_VERSION}-py3-none-any.whl "${PIP_WHL_URL}"; \
+     python3 - <<'PY'
+ import zipfile, os, glob, shutil, pathlib, subprocess, sys
+
+ wheel = '/tmp/pip-24.1.2-py3-none-any.whl'
+ dest = '/google-cloud-sdk/platform/bundledpythonunix/lib/python3.11/site-packages'
+
+ # Extract the new pip wheel into site-packages to replace runtime modules
+ with zipfile.ZipFile(wheel) as z:
+     z.extractall(dest)
+
+ # Also replace any bundled wheels used by ensurepip in both the runtime copy and
+ # the Google Cloud SDK installer backup directory reported by scanners
+ ensurepip_dir = pathlib.Path('/google-cloud-sdk/platform/bundledpythonunix/lib/python3.11/ensurepip/_bundled')
+ for f in ensurepip_dir.glob('pip-24.1-py3-none-any.whl'):
+     f.unlink()
+ shutil.copy('/tmp/pip-24.1.2-py3-none-any.whl', ensurepip_dir / 'pip-24.1.2-py3-none-any.whl')
+
+ backup_dir = pathlib.Path('/google-cloud-sdk/.install/.backup/platform/bundledpythonunix/lib/python3.11/ensurepip/_bundled')
+ if backup_dir.exists():
+     for f in backup_dir.glob('pip-24.1-py3-none-any.whl'):
+         f.unlink()
+     shutil.copy('/tmp/pip-24.1.2-py3-none-any.whl', backup_dir / 'pip-24.1.2-py3-none-any.whl')
+
+ # Cleanup downloaded wheel
+ os.remove(wheel)
+ PY
+ # ------------------------------------------------------
+
```

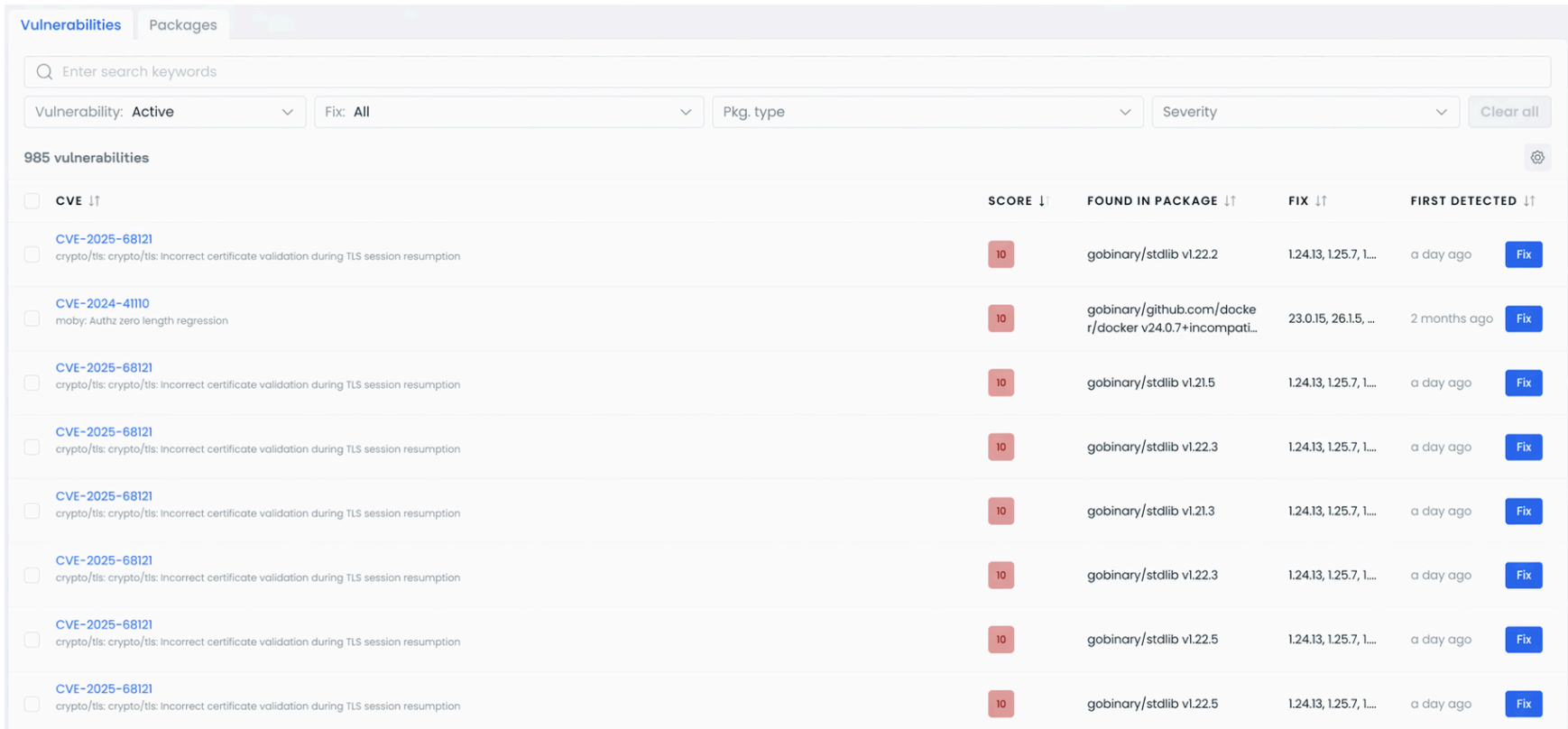
Attempt #2

Consolidate logic into a self-orchestrating single AI agent with multiple tools

```
for image in vulnerable_images:
    result = agent.run(prompt=f"""
    You are a security engineer tasked with remediating security vulnerabilities in a container image.

    You are given a git repo that contains the source code of a container image: {image_name}
    You are tasked with fixing the following vulnerabilities: {vulnerabilities}

    Use the provided tools to explore the repo and find relevant files
    (Dockerfiles, application dependency files, etc.)

    Once you are done fixing, try to build the image to make sure you haven't broken anything.
    If building fails because of your changes, adapt them until the build is succesfull.
    If building fails for unrelated reasons, continue without building.

    If you're able to build the image, scan it to make sure the vulnerabilities are fixed.
""", tools=[
    read_file_tool,
    build_tool,
    scan_image_tool,
    open_pr_tool, ... # and more tools
], context={
    'image_name': image.name,
```

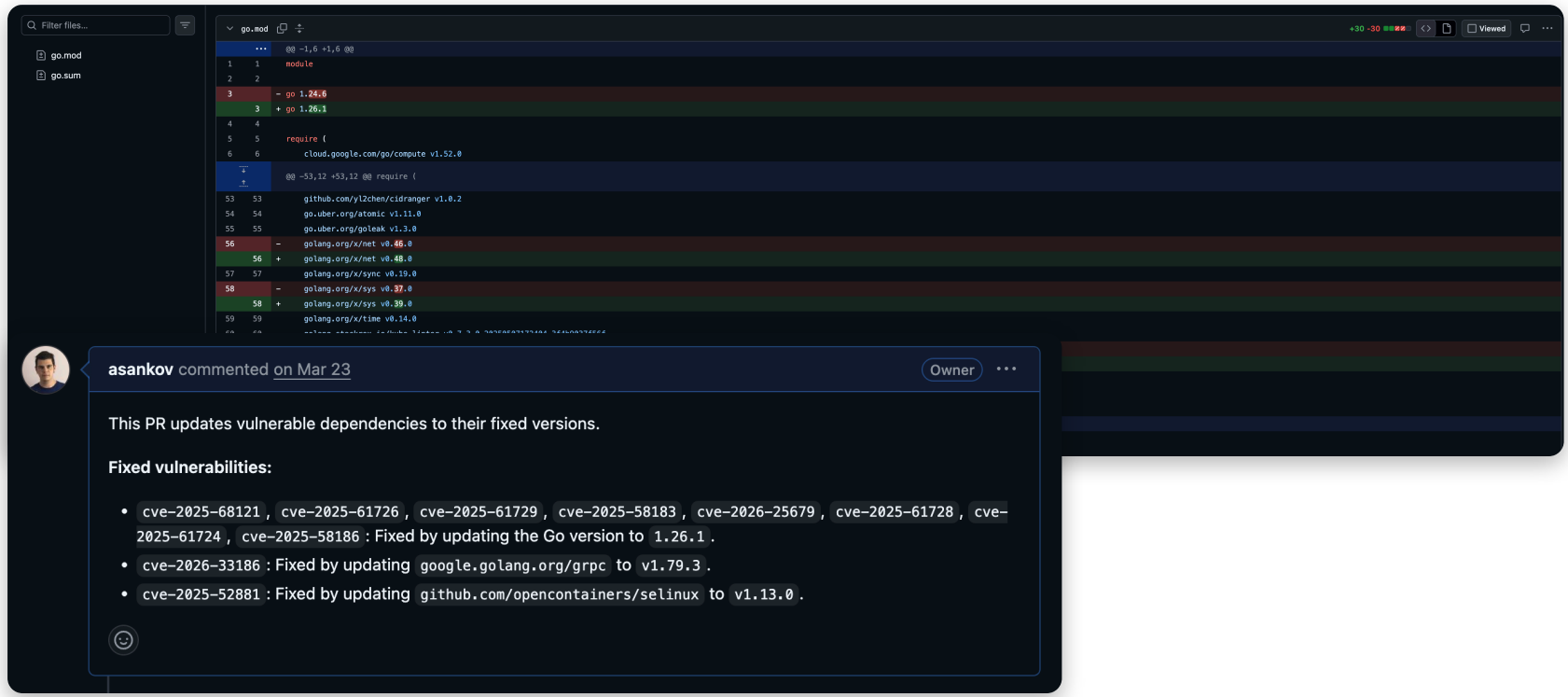
Attempt #2

Gave more control to the users



The Result

Small, boring … but mergeable PRs



Attempt #2

Why this worked better

Mistake #1: LLMs tend to overdo things, especially when given tasks that are not properly defined and scoped
Solution: Define the task better and limit the context given to the LLM

Mistake #2: We were doing stuff which was not mandatory and could have been done on a best-effort basis
Solution: Recover from failures such as not being able to find a file or a failing build and continue executing the task

Mistake #3: We were treating all images as equal
Solution: Give the user control over which images to fix

Attempt #2

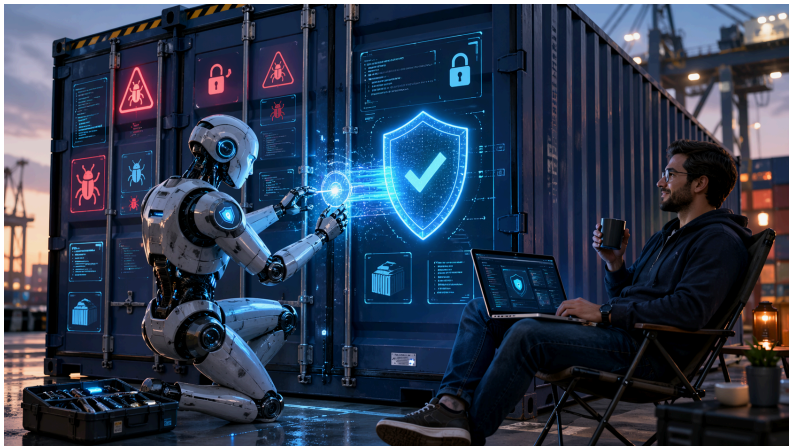
Lessons Learned

LLMs overthink and overdo stuff, but they can be prompted into doing the right thing.

LLMs are resourceful and can find their way out of a problem if given enough freedom.

This problem is as much technical as it is organizational.

Will AI Replace Us?



Thank you!

Questions?

Anton Sankov

asankov.dev

Download the slides

asankov.dev/teaching-ai-to-fix-vulnerabilities